IT2901 - Informatikk prosjektarbeid II

μC Software Store

Project Report

09_Arduino

Jeppe Eriksen, Bjørn Arve Fossum, Ståle Semb Hauknes,
Wilhelm Walberg Schive, Nina Margrethe Smørsgård, Robin Tordly

May 12, 2013

Abstract

In cooperation with IDI at NTNU, SINTEF was the contracting customer for this project, hiring the group consisting of six students on the course IT2901.

The project's aim was to expand the usability of existing Arduino devices, and simplify the process of installing new functionality. This report is an overview of the project of developing the system to do so, and how the work of the developer team evolved alongside the demands and needs of the customer. The base idea was to create an easily operable system running on Android based devices for installing compatible applications to the customers' existing Arduino device. This was done by implementing the STK-500v1 protocol for communication with Arduino devices, and creating a market application for Android. This application handles connection via Bluetooth and data transfer over that protocol, as well as containing the market for browsing of available apps.

The development of both the protocol and the Andriod application was done in Java, as both were designed to run on Android devices. Implementing and integrating the protocol proved to be among the most time consuming tasks in the project.

Contents

1 Introduction **1**
- 1.1 The course . . . 1
- 1.2 The group . . . 1
- 1.3 The customer . . . 2
- 1.4 Problem description . . . 2
- 1.5 The goal . . . 2
- 1.6 Definitions . . . 2
  - 1.6.1 Registered trademarks . . . 4

2 Prestudy **6**
- 2.1 Development methods . . . 6
  - 2.1.1 Rational unified process . . . 6
  - 2.1.2 Scrum . . . 7
  - 2.1.3 Waterfall model . . . 8
  - 2.1.4 Spiral model . . . 8
  - 2.1.5 Lean software development . . . 8
- 2.2 Existing solutions . . . 9
  - 2.2.1 Market applications . . . 9
  - 2.2.2 Pebble . . . 10
- 2.3 Over-the-air transfer . . . 11

3 Project management **13**
- 3.1 Development method . . . 13
  - 3.1.1 Changes to description . . . 13
- 3.2 Team roles . . . 14
  - 3.2.1 Role description . . . 14
  - 3.2.2 Division of labor . . . 15
- 3.3 Communication . . . 15
- 3.4 Project planning . . . 15
  - 3.4.1 Work breakdown structure (WBS) . . . 16
    - 3.4.1.1 Gantt chart . . . 16
    - 3.4.1.2 Work breakdown structure table (WBS table) . . . 17
    - 3.4.1.3 Work breakdown structure tree (WBS tree) . . . 19
  - 3.4.2 Editions to WBS . . . 20
- 3.5 Risk analysis . . . 20

4 Requirements **22**
- 4.1 Functional requirements . . . 22

4.2 Non-functional requirements . . . 23
4.3 Changes to requirements . . . 23
4.3.1 Removals . . . 23
4.3.2 Additions . . . 24
4.4 Use cases . . . 24

5 Development environment 35
5.1 Development tools . . . 35
5.1.1 Integrated development environments (IDEs) . . . 35
5.1.1.1 Eclipse . . . 35
5.1.1.2 Arduino IDE . . . 36
5.1.2 Codebase management and versioning . . . 36
5.1.2.1 Git version control software . . . 36
5.1.2.2 GitHub . . . 36
5.2 Project management tools . . . 36
5.2.1 Google Docs . . . 36
5.2.2 Microsoft Word . . . 37
5.2.3 LaTeX . . . 37
5.2.4 Google Calendar . . . 37
5.2.5 Dropbox . . . 37
5.2.6 TextMate and Sublime Text 2 . . . 37
5.2.7 Web sequence diagrams . . . 37
5.2.8 Wunderlist . . . 37
5.2.9 Doodle . . . 38
5.2.10 Fritzing . . . 38
5.2.11 OmniGraffle . . . 38
5.3 Test management . . . 38
5.3.1 Robotium . . . 38

6 Design and implementation 39
6.1 System architecture . . . 39
6.1.1 Browse shop . . . 43
6.1.2 STK500v1 . . . 44
6.2 Design of the Android application . . . 47
6.2.1 Design guide . . . 47
6.2.2 Design decisions . . . 54
6.2.3 Changes to the design . . . 55
6.3 Database implementation . . . 55
6.3.1 Database model . . . 56
6.3.2 Database details . . . 56
6.3.2.1 Table: app . . . 56
6.3.2.2 Table: binaryfiles . . . 57
6.3.2.3 Table: developer . . . 57
6.3.2.4 Table: requirements . . . 57
6.3.2.5 Table: appusespins . . . 57

7 Testing 58
7.1 Testing strategy . . . 58
7.1.1 Unit testing . . . 58
7.1.2 Integration testing . . . 58

7.1.3 Usability testing . . . 59
7.1.3.1 Tasks . . . 59
7.1.4 User acceptance testing . . . 59
7.2 System testing . . . 60
7.3 Test results . . . 63
7.3.1 Unit testing . . . 63
7.3.2 Integration testing . . . 63
7.3.3 System testing . . . 64
7.3.4 Usability testing . . . 64
7.3.5 Expert reviews . . . 64
7.3.6 User acceptance testing . . . 65
7.3.7 Summary . . . 65

8 Roadmap for further development 67
8.1 The μC Software Store application . . . 67
8.1.1 Database . . . 67
8.1.1.1 Uploading interface . . . 67
8.1.2 Standard for defining μC Software Store compatible microcontroller-based devices . . . 68
8.1.3 Performance . . . 68
8.1.4 GUI . . . 68
8.2 STK500 and over-the-air installation . . . 68
8.2.1 Hard reset . . . 68
8.2.1.1 How to hard reset . . . 68
8.2.2 Security . . . 69
8.2.3 Protocols . . . 70

9 Evaluation 71
9.1 Challenges . . . 71
9.1.1 Licensing . . . 71
9.1.2 Protocol . . . 71
9.1.3 Hard reset . . . 72
9.2 Interaction with the customer . . . 73
9.3 Group interaction . . . 73
9.4 Lessons learned . . . 74
9.4.1 Prestudy . . . 74
9.4.2 Project management . . . 74
9.4.3 Design and Implementation . . . 75
9.4.4 Testing . . . 75
9.5 Conclusion . . . 75

List of Tables 76

List of Figures 76

Bibliography 78

Appendixes 81

A Project documents 81
A.1 Status report example . . . 81

A.2 Activity plan example 82

B Iterations **83**
B.1 Iteration 1 83
B.1.1 Summary 83
B.1.2 Overview 83
B.2 Iteration 2 83
B.2.1 Summary 83
B.2.2 Overview 84
B.3 Iteration 3 84
B.3.1 Summary 84
B.3.2 Overview 84
B.4 Iteration 4 84
B.4.1 Summary 84
B.4.2 Overview 84
B.5 Iteration 5 85
B.5.1 Summary 85
B.5.2 Overview 85
B.6 Iteration 6 85
B.6.1 Summary 85
B.6.2 Overview 85
B.7 Iteration 7 85
B.7.1 Summary 85
B.7.2 Overview 86

C Standard for defining μC Software Store compatible microcontroller-based devices **87**
C.1 Introduction 87
C.2 Basics 87
C.3 Technical 88
C.4 Example 89

D μCSS User manual **91**
D.1 Introduction 91
D.2 Welcome Screen 92
D.3 Device List 93
D.3.1 Adding Device 94
D.3.1.1 QR Code 94
D.3.1.2 Serial 94
D.4 The Shop 95
D.4.1 Searching 95
D.5 App View 96
D.6 Settings 97

E Installation guide **98**
E.1 Upload the first application 98
E.1.1 Arduino IDE 98
E.1.2 Configure Bluetooth module 98
E.1.3 Supported Bluetooth modules 100
E.1.3.1 RN-42 100

E.1.3.2 HC-05 100
E.1.4 Upload supported application 101
E.2 Android application 101
E.3 Recover 101
E.4 Develop your own apps 101

1 Introduction

This document is the report required by the course IT2901 at the Norwegian University of Science and Technology (NTNU), written during spring 2013. The goal of this report is to give the reader an overview of the project. The task was to develop a market application on Android for Arduino programs, and installing those applications over Bluetooth. For users looking for help to use the resulting software, see the installation guide (Appendix E) or the user manual (Appendix D).

1.1 The course

IT2901 Informatikk Prosjektarbeid II is the course in which this project was carried out. It is commonly taken the last semester of a bachelor in informatics. This report is written for the customer, supervisor and teachers of this course.

1.2 The group

The group consisted of six students from Computer Science at NTNU, each possessing different technical skills. All were third year students in the Informatics Bachelor program.

Bjørn Arve Fossum
Experience with Java, MySQL, PHP and C#.

Jeppe Eriksen
Experience with Java, MySQL, and basic Android development.

Nina Margrethe Smørsgård
Experience with Java, MySQL, LaTeX, and basic Android development.

Robin Tordly
Experience with Java, Android, MySQL, PHP and SQLite.

Ståle Semb Hauknes
Experience with Java, MySQL, PHP and Arduino.

Wilhelm Walberg Schive
Experience with Java, MySQL, and the basics of Arduino project development.

1.3 The customer

SINTEF is the largest independent research organization in Scandinavia. It is a non-commercial organization with close ties to NTNU and international activity with clients in 60 different countries. There is a joint effort by NTNU and SINTEF regarding lab work, many are employed by both and there are labs funded by both companies in order to facilitate continued research.

Throughout the project, customer interraction was done through Babak Farshchian from SINTEF.

(a) (b)

Figure 1.1: Customer: 1.1(a) SINTEF, 1.1(b) NTNU.

1.4 Problem description

The project task was to develop a platform for easy use, setup and sharing of Physical User Interfaces (PUIs) for Arduino. The task was divided in three main parts: a market application, over the air installation and example PUIs.

The purpose of the market application was to allow users of Arduino to browse and download PUI applications for their Arduino board on a mobile Android device. This market application was to be a simplified version of e.g. Google Play, where users easily can browse and install whatever applications they desire. An application selected for installation in the market application should be prepared on the mobile device, and installed over the air on an Arduino board using Bluetooth. The example PUIs were mostly intended to demonstrate the feasibility of the finished product.

1.5 The goal

The goal of the project task was to make Arduino easier to use for ordinary people, by allowing easy browsing, sharing, and installation of applications on Arduino boards. By developing an application for over the air installation of applications, the finished product should ease the process of both installing and updating PUIs on an Arduino.

1.6 Definitions

This is a list of terms and abbreviations used throughout the project report in order to clarify and explain their meaning.

μCSS
μC refers to microcontroller, and μCSS refers to Microcontroller Software Store, which is the name of the Android application developed.

Activity
In Android development, an activity provides a user interface for a single screen in your application.

Action overflow
In Android development, actions that does not fit in the action bar (menu bar on the top) are moved automatically to the action overflow.

Android:
An operating system for mobile devices based on the Linux operating system. It is developed by Google and the Open Handset Alliance. Applications for Android devices are written in Java, and all the software is open source released under the Apache License.

Apache:
A software foundation focused on open source and community driven software.

Arduino:
A tool for making Physical User Interfaces (PUIs). It is an open-source physical computing platform based on a simple micro-controller board.

AVRDude:
This is a tool to upload programs to micro-controllers from the Unix command line. This software has a lot of features, such as reading and writing to the micro-controller's memory over a serial connection. This software can also compile your C code into an Intel Hex file, a file full of binary information, that can be sent to the Arduino with the STK500v1 protocol. Normally this is done with a cable between a computer and the microcontroller.

Baud rate:
Baud rate refers to the number of signal or symbol changes that occur per second. A symbol is one of several voltage, frequency, or phase changes [1].

Bootloader
A bootloader is a computer program that loads the main operating system or runtime environment for the computer after completion of the self-tests.

Commit
In relation to version control software, a commit is the act of recording changes to repository.

Content provider:
Standard way for Android applications to share information between applications and services.

Customer:
Refers to SINTEF with Babak Farshchian as their representative.

Lesser GPL
GNU public licence is a licence geared towards supporting open source software. The lesser additions (Lesser GPL) also allows for inclusion in proprietary software.

JUnit:
A unit testing framework specifically written for Java.

Intent
In Android development, an intent is an abstract description of an operation to be performed.

Over-the-air Refers to wireless transfer of data over Bluetooth.

PUI:
An acronym for Physical User Interface. A PUI is a user interface which interacts with digital information through the physical environment.

Repository
A structured and indexed storage location for code.

STK500:
This is a standard protocol used by many microcontrollers to upload and download memory, including the Arduino.

Sync adapter:
A link between the content provider and server, updating either of the two to be in sync with the other.

Version control software (VCS)
Software for maintaining and keeping track of different versions of content.

Wrapper:
A coding pattern which handles reading availability and blocking.

1.6.1 Registered trademarks

Android™
Android is a trademark of Google Inc.
The Android robot is reproduced or modified from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.

Apache™
Apache is a trademark and service mark by ASF Projects for community developed open source software products.

Arduino™
Arduino is a trademark of Arduino team.

Atmel®
Atmel, AVR®061 and AVR®062 are registered trademarks or trademarks of Atmel Corporation or its subsidiaries, in the US and/or other countries.

Bluetooth®
The Bluetooth word mark, figure mark, combination mark, and Bluetooth Smart and Bluetooth Smart Ready marks are all trademarks that are owned by the Bluetooth SIG and licensed out for use to companies that are incorporating Bluetooth wireless technology into their products.

Doodle™
Doodle is a trademark of Doodle AG.

Dropbox™
Dropbox is a trademark of Dropbox.

Eclipse™
Eclipse is a trademark of the Eclipse Foundation.

Fritzing™
Fritzing is a trademark of the Friends-of-Fritzing e.V.

Google™
Google, Google Calendar™, Google Docs™ and Google Play™ are trademarks of Google Inc.

Microsoft®
Microsoft and Microsoft Word® are trademarks of Microsoft Corp.

OmniGraffle™
OmniGraffle is a trademark of Apple Inc.

Raspberry Pi®
Raspberry Pi is a trademark of the Raspberry Pi Foundation.

TexMate™
TextMate is a trademark of Allan Odgaard.

SQLite™
SQLite is a trademark of Hwaci.

QR Code®
QR Code is a registered trademark of Denso Wave.

2 | Prestudy

A thorough study of the problem area, existing solutions, development methods etc. is important to gain the knowledge needed to make good decisions concerning the project. In this chapter the research performed in the first phase of the project will be presented.

2.1 Development methods

Several development methods were examined in the beginning of the project to ensure that a suitable model was chosen. In the following section the research performed on different development methods will be presented. Regarding the choice of development method, the decision of the group will be presented in Chapter 3.1.

2.1.1 Rational unified process

Rational Unified Process (RUP) is an iterative and incremental software development process model. It is a process model that aims to capture the best practices in modern software development and present them in a tailorable form [2]. Each iteration in this model results in an increment, which is a release of a prototype and an improvement of the previous iteration. Most of the iterations will, in addition to work on prototypes, also contain work on requirements, design, implementation and testing.

A feature of this model is that it is use case driven. Every iteration takes a set of use case scenarios from the requirements and use those for the content of the iteration. The model also requires the team to focus on the critical risks of the project early in the development process. This ensures that problem areas and uncertainties is dealt with before severe problems arise. An illustration of the model is shown in Figure 2.1.
The model consists of four phases:

- Inception
- Elaboration
- Construction
- Transition

Inception phase The inception phase is the smallest phase, and should cover the work on identifying risks, creating use cases, establishing boundaries and so on. Cost estimates are calculated in the inception phase. This phase should result in a document that states

the core of the product, with a preliminary overview of the architecture, requirements, use cases and risks.

Elaboration phase In the Elaboration phase the team is expected to filter out the majority of the system requirements and validate the system architecture. A detailed overview of the product should be established in this phase, and the project plan should be developed. The documentation produced in this phase is essential for the work done in the Construction phase.

Construction phase The Construction phase is the largest phase in the model. In this phase all the remaining features of the product are developed and integrated, and thoroughly tested. At the end of this phase, the finished product should be ready to be presented to the customer.

Transition phase The Transition phase is where the system is deployed to the target users. Feedback from the customer might result in further refinements, and new versions of the product. Elements in this phase are beta testing and training of the users.

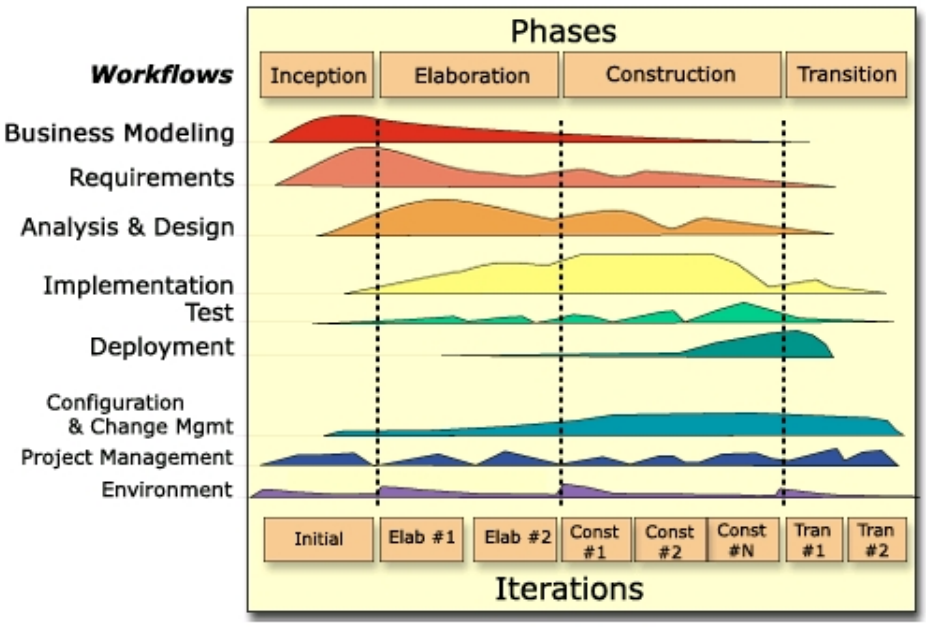


Figure 2.1: Illustration of the Rational Unified Process. The waves indicates where the focus and work is taking place in each phase.

2.1.2 Scrum

Scrum is an iterative and incremental development framework for development of complex information systems. When using Scrum, the development of the product is developed in small pieces, with each piece building upon previously created pieces. The use of Scrum requires the team to be divided into specific roles, where each role has its own responsibility. The following are the core roles of the Scrum team [3] [4]:

Product owner: The product owner represents the stakeholders in the project, and is the voice of the customer. It is the product owner's responsibility to ensure that the Scrum team at all times is working with the right things seen from a business perspective.

Scrum master: The scrum master should act as a buffer between the development team and distracting influences, so that the development team can deliver potentially shippable products at the end of each sprint. The scrum master should keep the development team focused at all times.

Development team: The development team is made up from three to nine persons with cross-functional skills. This team does all the actual work, including development, testing, designing and so on.

2.1.3 Waterfall model

The waterfall model is among the first process models to be introduced. In this model, each phase in the development must be completed before one can advance to the next phase. At the end of each phase in the development, a review takes place. In this review it is determined if the project is on the right path and if it should continue or not. The phases in the development do not overlap [4].

2.1.4 Spiral model

The spiral model is a software development model intended for large and complicated projects. It combines elements from the waterfall model and prototyping models, and uses an iterative approach. Based on this it allows for incremental releases of the product, as prototypes [4].

2.1.5 Lean software development

Lean is an agile software development methodology that is defined by seven principles [5]:

Eliminate waste. Everything that does not add any value to the customer is considered a waste.

Amplify learning. The development environment should the designed so it amplifies the learning process of the developers. Defects should be prevented by running tests as soon as the code is written.

Decide as late as possible. By delaying decisions as much as possible, better results can be achieved. There is always uncertainty associated with software development, and delaying options as much as possible results in more flexibility later.

Deliver as fast as possible. The sooner the product is delivered, the sooner feedback can be received.

Empower the team. The managers are taught how to listen to the developers, so they can explain better what actions could be taken and give suggestions for improvements.

Build integrity. The customer needs to have influence and inspection of the project, and an overall experience of the system to be developed.

See the whole. Decompose the system into smaller parts and find and eliminate defects.

2.2 Existing solutions

This is a summary of existing solutions similar to the project assignment. This section is divided in two subsections; one for the market application and one for the over-the-air transfer. The existing products were evaluated on the following criteria:

- To what degree the product fits the assignment.
- Can the product, or parts of it, be reused for the assignment? Can it lead to licensing issues?

2.2.1 Market applications

UbiBazaar is a prior project which created an universal app store in PHP. This project had potential for serving as a back end, but this was later discarded. The idea of a website for uploading applications was the primary idea for the project. Upon revising the project it was deemed too unstable and incomplete to properly function for our intended use. The project used an outdated version of PHP which caused numerous bugs which would take up too much time to attempt to fix properly.

It did have a way for developers to follow each other, which would help creating a community for the developers. For further development of μCSS, UbiBazaar would also be able to support the development of applications aimed at, for example, Raspberry Pi. This was not included as the goal of our project and therefore discarded.

Google Play is the market application for Android provided by Google. It is an application that allows for easy browsing and installation of applications on Android devices. It detects the model of the phone in use, and only shows applications supported by that phone. This functionality is similar to the requirements for the Android application in the project. Since Android users most likely are used to this user interface, shown in Figure 2.2, it was desirable to emulate this user interface.

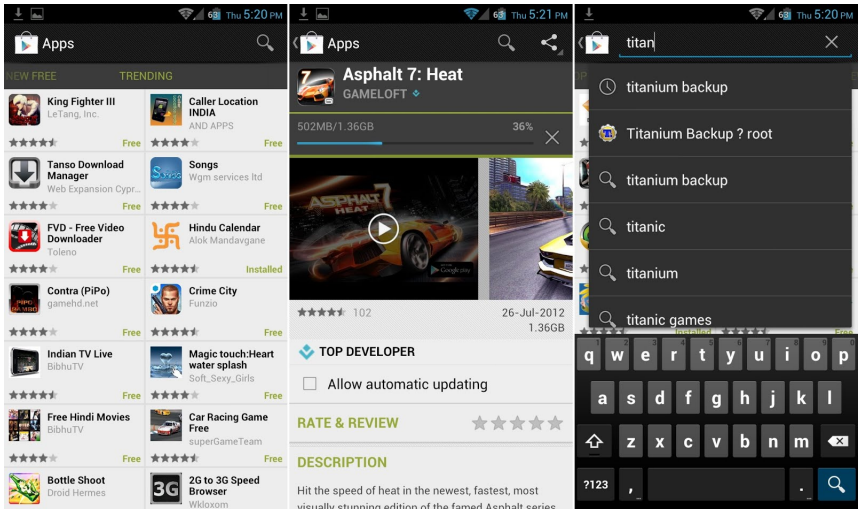


Figure 2.2: Screenshots from the Google Play Store showing their graphical user interface throughout the application.

Google Play fits the assignment in the way that it is a market applications where one can download applications. This was useful for the development of the product, since it was possible to use the same principles in the assignment. Google Play is not open source and could therefore only be used as a source of inspiration for the project. It was not possible to reuse the code or other parts of the application in the development.

2.2.2 Pebble

Pebble is a watch, shown in Figure 2.3, that offers over-the-air transfer of applications. The main differences between this watch and a standard Arduino, is that Pebble have written a custom operating system, and contains different hardware than an Arduino. Pebble is also using a microcontroller that is significantly faster and contains a lot more memory, which means that it is much easier to develop an operating system for it.

To transfer a new application to the watch, the user connects the mobile phone to the watch using Bluetooth. The user can then open the Pebble application store and upload applications made for this watch.

Figure 2.3: Pebble Watches with different applications changing the watch-faces.

The principles are quite similar to what we planned to develop, but this platform is not made to expand to other platforms. There was very little documentation on the Pebble website, so the potential for reuse in this project appears minimal.

2.3 Over-the-air transfer

Usually when you program the Arduino you use a computer with a USB cable connected to the Arduino board. USB is used because it offers a stable connection and a standardized cable. This project's main goal was to use Bluetooth instead of a wired connection to program an Arduino device. To connect to the Arduino via Bluetooth it was wired as shown in Figure 2.4.

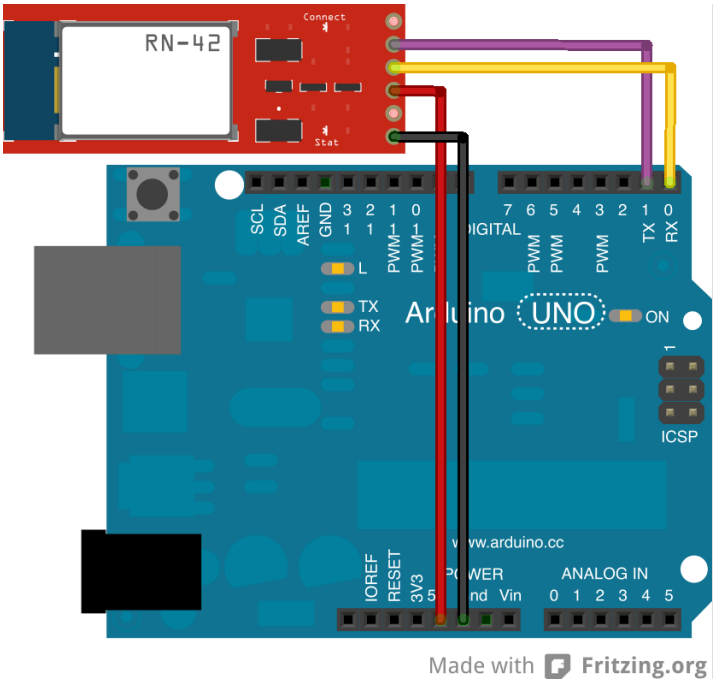


Figure 2.4: How a Bluetooth piece is connected on an Arduino device.

An extended prestudy concluded with the Bluetooth module RN-42 [6] being a well documented and promising component for the purposes of over-the-air installation. The protocol to upload new code to the microcontroller is quite advanced. The uploading must be done without any errors and at the correct baud rate.

3 | Project management

The following sections details the project management decisions of the project. This includes choice of development method, team member responsibilities, communication channels and risk analysis.

3.1 Development method

Based on the customer requirement of short iterations, it was decided early on to adapt an iterative and incremental development method for the project. By working in an iterative manner, it was possible to present prototypes and work done to the customer every week, and at the same time receive feedback. In this fashion it was possible for the customer to continuously present his thoughts on the product, and propose changes where he thought it was necessary. The iterative approach on the project also made it easier for the group to set deadlines and milestones for specific parts of the product. A summary and overview of each iteration is described in appendix B

All the members of the group have taken the course TDT4140 - Systemutvikling, and therefore have some knowledge about different development methods. The Rational Unified Process (RUP) model, as described in Chapter 2, is an extensible framework [2] that met the requirements the group had for a process model. RUP was chosen as process model for the project above the other models described in Chapter 2 as it best met the needs of the group. The fact that the model was tailorable and divided the entire development process into phases that were easy to adapt to, made it preferable to the other models examined. It was also appealing that the model required the team to focus on critical risks early in the development to avoid surprises.

3.1.1 Changes to description

As mentioned in the previous section, it was clear that the use of an iterative model was in order. The specific documentation of the development method was deferred for three weeks, however. This was due to the uncertain future of the project (over-the-air installation was said to potentially be impossible) and the need to get the first iteration presented to the customer; precisely documenting the development method was deemed to be wasteful at that point in time. The development method description was also refined over the course of the project.

It was decided to adopt RUP as process model, though with some minor changes. First of all, the Transition phase of the model was deemed unnecessary. This was mainly due to the lack of time and resources for beta testing and new versions of the product. This phase was instead replaced by the closeout phase, described in chapter 3.4. Further, no cost estimates were developed for the project. Because no money was involved in the production of the product, this was also deemed unnecessary. Time and resource estimates, however, were established during the Inception phase. Lastly, due to lack of time and experience, testing was less prioritized than the implementation.

In addition to the changes mentioned above, use of Lean principles were adapted from the start of the project. As described in the section about Lean Software Development, this methodology is defined by seven principles. Of these, especially *Eliminate waste* and *Build integrity* were used actively by the team during the development of the product and project report. These principles is described in chapter 2.1.5.

3.2 Team roles

The group was organized in different roles based on skill and experience. Each team member was given a responsibility for some code-packages. Further, the team was divided into six subgroups where each member was held responsible for his or her group. These groups were respectively: group leader, documentation and substitute leader, Android and GUI, Arduino and PUI, over-the-air and test. The distribution of team roles are listed in Table 3.1

3.2.1 Role description

The division of the group was an important feature. Every member knew who to contact about a specific problem or task.

Group leader was responsible for the progress in the overall project. This person ensured progress and priorities for deadlines.

Documentation and substitute leader was responsible for management of documentation and reports. In absence of the group leader, this person took on the group leader's responsibilities. This person was also responsible for contact with the customer and supervisor.

Android and GUI developer was responsible for the Android development of the project.

Arduino and PUI developer was responsible for the Arduino part of the project. This implies contacting the Arduino-lab, requisitions for hardware, the coding part and over-the-air installation. This role was also responsible for development of the PUI examples.

Over-the-air developer was responsible for programming the Arduino over the air. This person was also responsible for making the first prototype with a Bluetooth connection.

Test developer was responsible for developing and executing tests for the complete project.

Table 3.1: Roles

Name	Role
Jeppe Benterud Eriksen	Group leader
Nina Margrethe Smørsgård	Documentation and substitute leader
Robin Tordly	Android and GUI developer
Bjørn Arve Fossum	Arduino and PUI developer
Ståle Semb Hauknes	Over-the-air developer
Wilhelm Walberg Schive	Test developer

3.2.2 Division of labor

The division of labor between the members of the group was largely done through the use of issues on GitHub (Chapter 5.1.2.2). Whenever a task arose it was created as an issue on this site. This way, all the members could see what needed to be done, assign different issues to themselves, see what others were working on, and therefore choose an appropriate issue to start working on.

Throughout the project the main consensus of the group has been to meet every day as long as there is not something else interfering. This has caused the group to work together almost every day and thus if a problem was encountered it could be dealt with quickly. Usually these meeting started at 09:00-10:00 and ended around 16:00. Any and all work done by anyone was registered in a document to ensure correct activity plans. Example of an activity plan is documented in Appendix A along with an example of a status report.

3.3 Communication

Most of the communication within the group was done at group meetings and when the group was working together. For communication between group members outside the meetings it was decided that the group should only use email and Skype. This was decided to avoid the confusion that might arise from using numerous channels of communication. Mobile phone was also used when immediate contact was necessary.

Communication between the group and the customer was mainly done during meetings or by email. The same applies for communication with the supervisor. The customer also had access to the entire project on GitHub. This meant that he could check up on the progress of the project at any time, and add issues or concerns directly into GitHub.

3.4 Project planning

In this section the planning of the project will be presented. The project was divided into five large milestones to represent the different phases of the project. The phases of the Rational Unified Process (RUP) are not explicitly stated in the work breakdown structure, as some of the milestones stretches over several of the phases in the process model.

In the work breakdown structure (WBS), the first part of the Project Management activity and the Planning activity represents the inception and elaboration phase of RUP. Further, the Execution and Testing activities represents the construction phase. As mentioned above, the transition phase of RUP was deemed unnecessary. It was expected that tasks related to project management would appear at least until the technical part of the project was concluded, and it was therefore decided to let this activity run through the larger part of the project. In the closeout phase work on the final report was to be prioritized.

3.4.1 Work breakdown structure (WBS)

The WBS is a view on what work packages the project encompasses. It helps with communicating the work and processes to easily execute the project. The duration time show how much estimated time one task requires, and gives an assessment on how much effort is needed to complete the task.

3.4.1.1 Gantt chart

The Gantt chart in Figure 3.1 is showing the initial work breakdown structure created in the project.

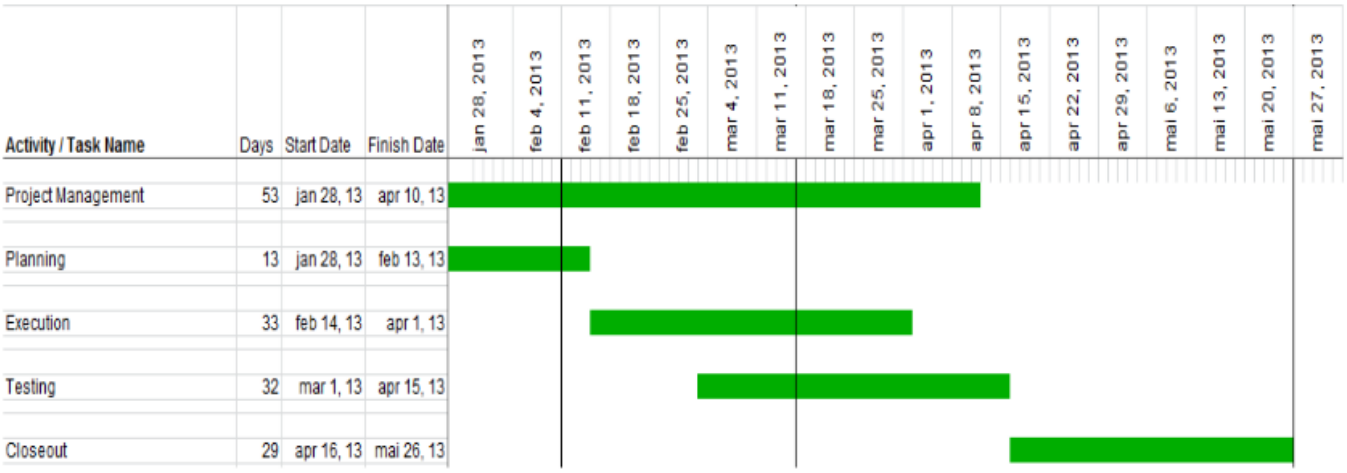


Figure 3.1: Gantt chart. Milestones are marked using vertical lines

3.4.1.2 Work breakdown structure table (WBS table)

Table 3.2 is showing a detailed overview of all the tasks to be preformed in each of the milestones. The WBS code of each task is used to relate it to the WBS tree shown below. This table also contains a description of each task.

Table 3.2: Work Breakdown Structure

WBS Code	Element name	Definition
1	μC Software Store	All work to implement an application store for Arduinos and over the air installation with two example PUIs
1.1	Project management	All the work in connection with project management
1.1.1	Meetings with customer	Determine the project status and decide on requirements and changes
1.1.2	Demonstration of work done for customer	Project team presents finished work for the customer
1.1.3	Risk management	Creating risk analysis and defining risks regarding the project and group
1.1.4	Status report	Write the status report
1.2	Planning	Work regarding the planning of different parts of the project
1.2.1	Requirements specification	Defining requirements and specifications
1.2.2	Determine Project team	Define roles and distribute tasks
1.2.3	Supervisor meeting	Meeting with supervisor for instruction and guidance
1.2.4	Develop project plan and use cases	Create use cases for scenarios
1.2.5	Research	Research areas where needed
1.3	Execution	Programming and execution of the project
1.3.1	Project kickoff meeting	Meeting with customer to present project plan
1.3.2	Verify & validate user requirements	Determine whether the group is in tune with the customers vision of the project requirements
1.3.3	Design GUI	Design the GUI using drawings and use cases displaying the different scenarios in the application
1.3.4	Create GUI	Create first version of the GUI designed in 1.3.3
1.3.5	Produce Software	Create the rest of the software
1.3.5.1	GUI and logic of μCSS	Implement GUI and create sorting logic

Continued on next page

Continued from previous page

WBS Code	Element name	Definition
1.3.5.2	Develop uploader application	Develop the application that will upload applications from μCSS to an Arduino device
1.3.6	Bluetooth implementation	Implement over the air installation of an Arduino device
1.3.6.1	Manage connection	Connect and maintain an Bluetooth connection
1.3.7	Documentation	Create all the necessary documentation needed for the report
1.4	Testing	Design and create test cases
1.4.1	Develop unit tests	Develop unit tests for the program
1.4.2	User testing	Develop user tests
1.5	Closeout	Work on final report

3.4.1.3 Work breakdown structure tree (WBS tree)

This is the WBS tree made for this specific project.

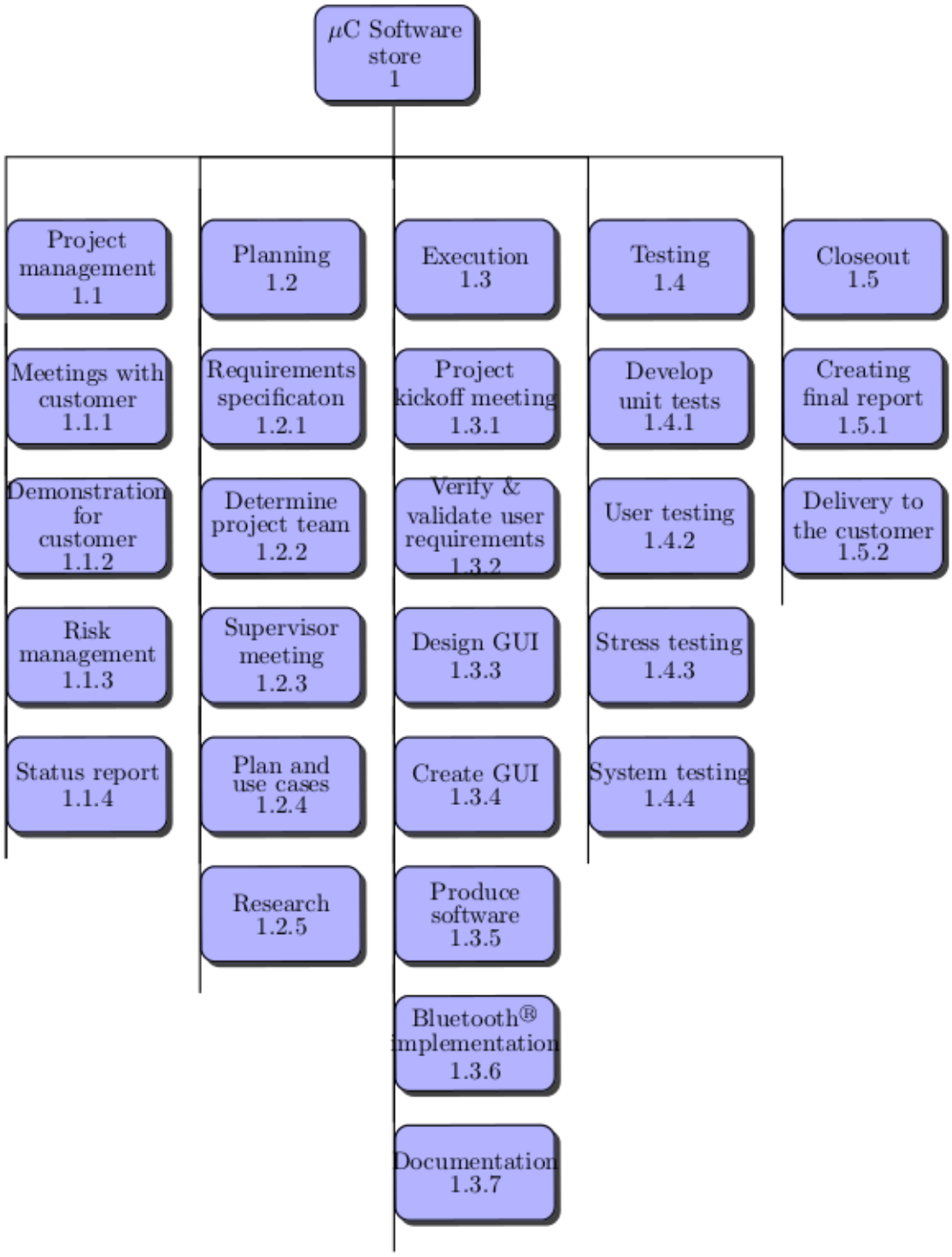


Figure 3.2: WBS tree showing a more detailed overview of the different tasks in each of the five milestones.

3.4.2 Editions to WBS

Due to challenges and unexpected work that emerged during the project, it was not possible to follow the WBS completely. First of all, a lot of unplanned work arose in connection with the implementation of STK500v1 in Java. The result of this was that not all group members were able to move to the closeout phase at the same time, as there still was unfinished work on the protocol.

3.5 Risk analysis

The risk analysis as shown in Table 3.4 states risks identified by the team. The importance of a risk is calculated by multiplying likelihood and impact. Higher numbers means higher importance for the project.

Table 3.4: Risk analysis

Description	Likelihood	Impact	Importance	Preventive Action	Remedial Action
Illness	7	2	14	Good communication and effective use of GitHub	Increase workhours and exchange tasks and responsibilities
Project complexity	6	5	30	Don't take on too much work	Cut down the demands
Customer issues	1	5	5	Agreement with customer and weekly feedback from customer	Use the original requirement specification
License incompability	7	7	49	Avoid integrating components with incopatible licenses	Discover other implementations or implment from scratch
Group conflicts or disagreements	3	3	9	Keep close contact to avoid surprises. Leader takes action	Contact supervisor and make an appointment
Over the air complexity	8	8	64	Have multiple alternative solutions and keep close contact with customer	Detail what was attempted as well as why it couldn't be solved in the final report.

Continued on next page

Continued from previous page

Description	Likelihood	Impact	Importance	Preventive Action	Remedial Action
Personal matters	8	5	40	Not much preventative action can be taken	Keep in touch and stay updated. In case you still can do tasks, claim one and tell the others

4 | Requirements

This chapter describes the functional and non-functional requirements of the system, together with use cases. At the end of the chapter, Chapter 4.3, during the project will be described. The requirements of the project will be kept high level and vague, to facilitate making choices as late as possible.

4.1 Functional requirements

FR01 **Over-the-air installation**
The Android application and the Arduino device should communicate over Bluetooth and install an arbitrary application from the Arduino Store in a simple two step process.

FR02 **Easy to use interface**
The Android application should be easy to use and easy to understand. It should not be necessary to do anything on the Arduino except for starting it. On startup it should search for nearby Bluetooth connections with paired devices.

FR03 **Example PUIs**
To demonstrate the Arduino Store (on Android), over-the-air installation, and the application in action on an Arduino, the group should create at least two example PUIs

FR04 **Reliable Bluetooth connection with the Arduino**
There should be a connection with the Arduino at all times when its in range, and it should automatically try to connect to the last connected device when the Android application starts up.

FR05 **Hide/Show incompatible applications**
The Android application should be able to hide Arduino applications in μC Software Store which are incompatible with the connected Arduino device

FR06 **Different ways to connect the Android to the Arduino**
It should be possible to connect to the Arduino with QR Code, manual input of MAC address and from a list of discovered devices.

4.2 Non-functional requirements

NFR01 **Usability**
People of all ages should be able to understand how to use the Android application and install applications on an Arduino

NFR02 **Reliability**
The Android application Bluetooth connection should be stable

NFR03 **Open source**
The project is under European R&D project SOCIETIES. All source code related to the Android application should be open source under Apache 2.0 license

NFR04 **Platform compatibility**
μCSS should be compatible with Android 2.3 and newer versions

NFR05 **Extensibility**
It should be easy to add features and extend this product later. The system should therefore be modular to simplify further development.

4.3 Changes to requirements

It was discovered that the over-the-air transfer of PUI apps to an Arduino device proved to be more demanding than first anticipated. Initial research done by the group on this area showed that a Java implementation of AVRDude already existed. Further research on the implementation, however, showed that it could not be used as it required programs not available on Android to be installed on the device. After an extraordinary meeting with the customer it was decided to change the focus of the assignment.

The goal of the project remained consistent with previous statements. A prototype of the Android store application was completed, though not all functionality was implemented and tested. Further work on the Android application was temporarily put on halt. At this point the resources were primarily focused on implementing the STK500 protocol in Java to allow for programming of Arduino from an Android device. For challenges and issues regarding the implementation of the protocol, see Chapter 9.1.2. See below for concrete additions and removals of the requirements.

4.3.1 Removals

μCSS server component: In agreement with the customer it was decided to remove the server side component of the assignment from the requirements. This was decided to pool resources to other, more critical tasks.

Sync adapter: As the server component of the assignment was removed from the requirements, there was no need for the sync adapter.

Filtering of incompatible apps: Due to lack of time, it was in agreement with the customer decided to omit functional requirement FR04 - Hide/show incompatible applications. As the functionality for hiding incompatible apps already had been implemented in the GUI at this point, it was decided to keep it there to illustrate the functionality. In the final version of the application the “Hide incompatible” option hides the empty

illustration apps from the user.

A standard for microcontrollers was written. This standard describes how filtering could be implemented. A document of this standardization can be found in Appendix C.

4.3.2 Additions

STK500 protocol: As of midterm, implementing the STK500v1 protocol in Java was added to the requirements.

Standard for defining μC Software Store compatible microcontroller based devices: As FR04 - Hide/show incompatible applications was removed from the functional requirements, a proposal for an implementation of this requirement were to be specified instead. This document were to contain a standard for how this functionality could be designed and implemented in future work.

4.4 Use cases

The following illustrations and tables were the foundation for the iterations in the development process.

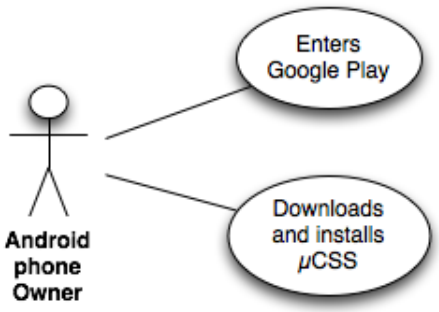


Figure 4.1: Use case 1: The user opens Google Play and finds and installs the μC Software Store application

The first use case describes how an Android-owner can browse Google Play to install the μC Software Store.

Table 4.1: Use case 1

ID	1
Name	Install μCSS
Goal	Have μCSS installed on the Android device
Actors	Android device owner
Precondition	None
Prerequisite	The actor has an Android device with Google Play
Main Flow	1. Opens Google Play 2. Searches for "μC Software Store" 3. Installs the application
Alternative Flow	None
Parent UC	None
Child UC	All

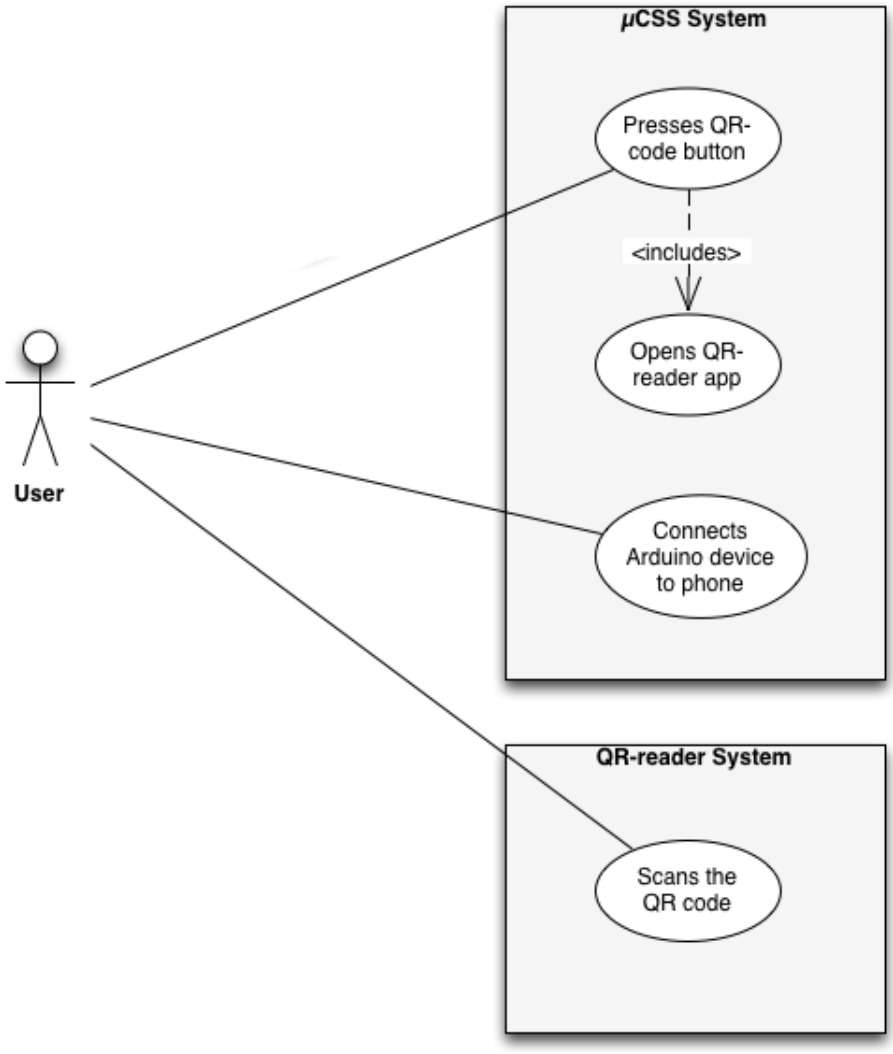


Figure 4.2: Use case 2: The user connects to the Arduino with QR code

The second use case describes how the user can connect to an Arduino device through the μC Software Store, by using a previously downloaded QR Code reader on an Android. The μCSS automatically detects available QR Code readers on the Android device, and prompts the user to choose a QR Code reader to complete the action. μCSS automatically detects the QR Code read by the included action, and determines whether or not this is a valid MAC address for a Bluetooth device. It then attempts to pair the Android with the Arduino. If there is no QR Code reader application installed on the Android, μCSS prompts the user to install one.

Table 4.2: Use case 2

ID	2
Name	Pair Arduino device and Android device with QR Code
Goal	Connect the Arduino device to the Android application with the use of QR Code
Actors	Arduino device owner
Precondition	Actor on "Add device" screen
Prerequisite	Installed μCSS Installed predefined QR-reader
End requirement	The Arduino device is connected to the phone via Bluetooth
Main flow	1. User presses the button indicating that he wants to pair with the Arduino device using QR Code 2. The system opens the QR reader application 3. The QR reader application reads the QR Code and returns the information it contains 4. The system pairs with the Arduino device
Alternative flow	2.a. The user does not have the a QR Code reader installed 2.b. The system prompts the user if he wants to install a QR Code reader 2.c. If no: stop 3.a. The QR reader is unable to read the QR Code 3.b. Try again or stop
Parent UC	1
Child UC	7

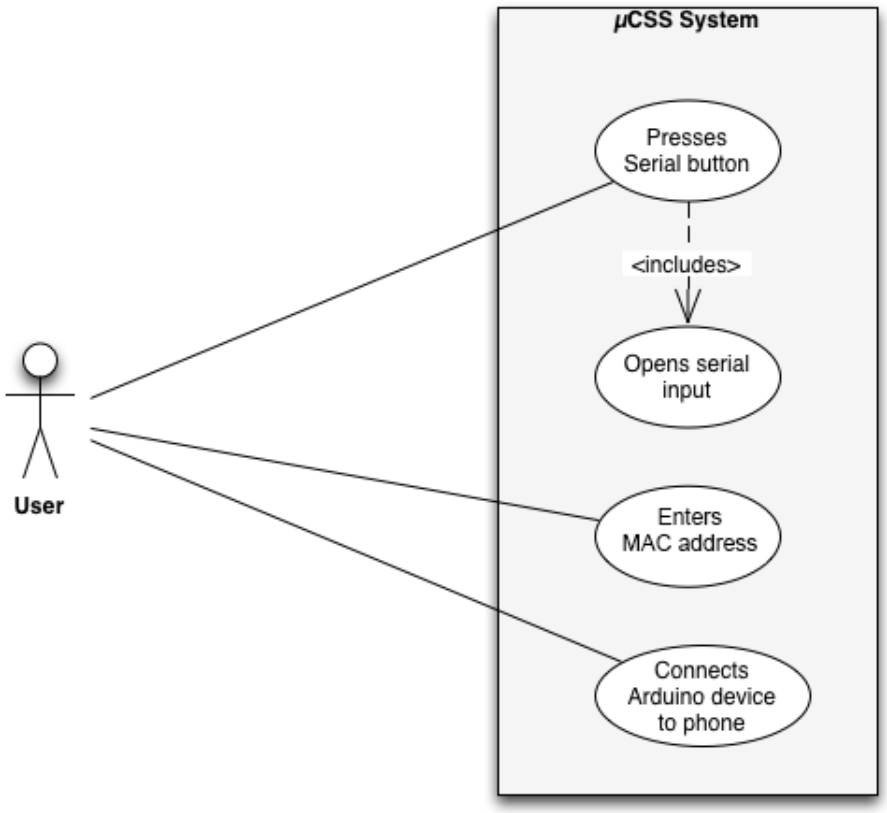


Figure 4.3: Use case 3: The user connects to the Arduino by manual input of MAC address

The third use case describes how the user can connect to an Arduino-device trough the μC Software Store, by inputting the MAC address of the device. μCSS determines whether or not this is a valid MAC address, and attempts to pair the Android with the Arduino. If the MAC address is invalid, μCSS prompts the user to retry input.

Table 4.3: Use case 3

ID	3
Name	Pair Arduino device with Android device manually entering MAC address
Goal	Connect the Arduino device to the Android application with the use of manual entering of MAC address
Actors	Arduino device owner
Precondition	Actor on "Add device" screen
Prerequisite	Installed μCSS
End requirement	The Arduino device is connected to the phone via Bluetooth
Main flow	1. User presses the button indicating that he wants to pair with the Arduino device using the MAC address 2. System opens dialog box for input 3. The user types the MAC address 4. The system pairs with the Arduino device
Alternative flow	3.a. The user misspells the MAC address 3.b. The system displays an error, and prompts again
Parent UC	1
Child UC	7

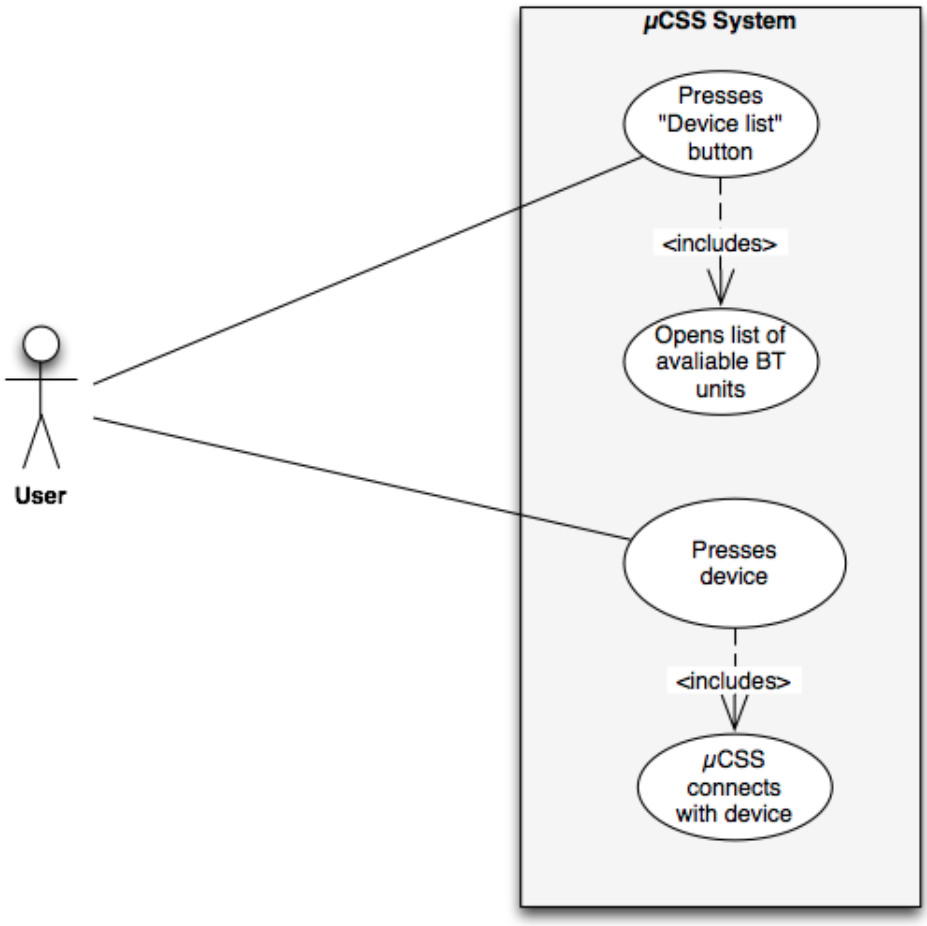


Figure 4.4: Use case 4: User uses the Bluetooth to scan for available connections.

The fourth use case describes how the user can choose an Arduino device from a list of available Bluetooth devices, and pair devices, from the menu in μCSS.

Table 4.4: Use case 4

ID	4
Name	Pair Arduino device with Android device using device list
Goal	Connect the Arduino device to the Android application with the use of device list
Actors	Arduino device owner
Precondition	Actor on "Devices" screen
Prerequisite	Installed μCSS
End requirement	The Arduino device is connected to the phone via Bluetooth
Main flow	1. User presses the device list button 2. System opens the list of available Bluetooth devices 3. The user presses the device to connect to 4. The system pairs with the Arduino device
Alternative flow	None
Parent UC	1
Child UC	7

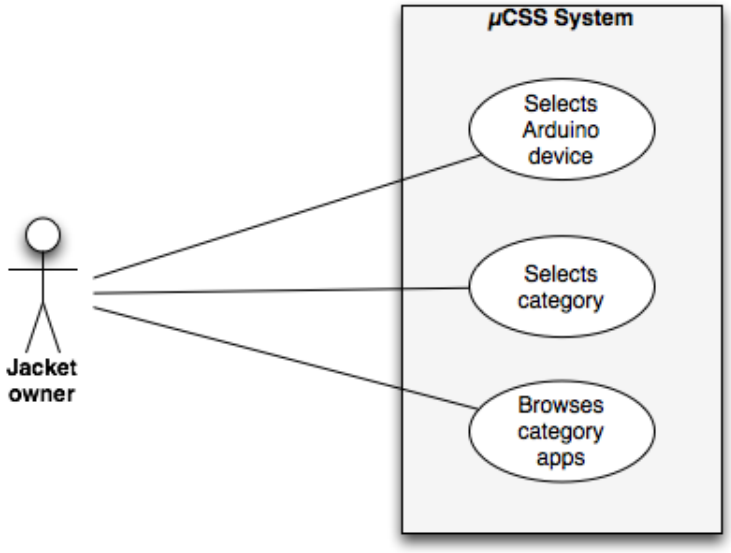


Figure 4.5: Use case 5: The user selects a category and browses the applications

The fifth use case describes how the user can browse apps in the μCSS, and filter them on categories. Browsing of apps can be done both with or without pairing of an Arduino device.

Table 4.5: Use case 5

ID	5
Name	Browse apps
Actors	Arduino device owner
Prerequisite	Installed μCSS
Precondition	Actor on "Device list"
End Requirement	None
Main Flow	1. The user selects an Arduino device 2. The user selects category (One category is named "all") 3. The user browses apps
Alternative Flow	1.a. The user does not select Arduino device, but browses anyway
Parent UC	1
Child UC	7

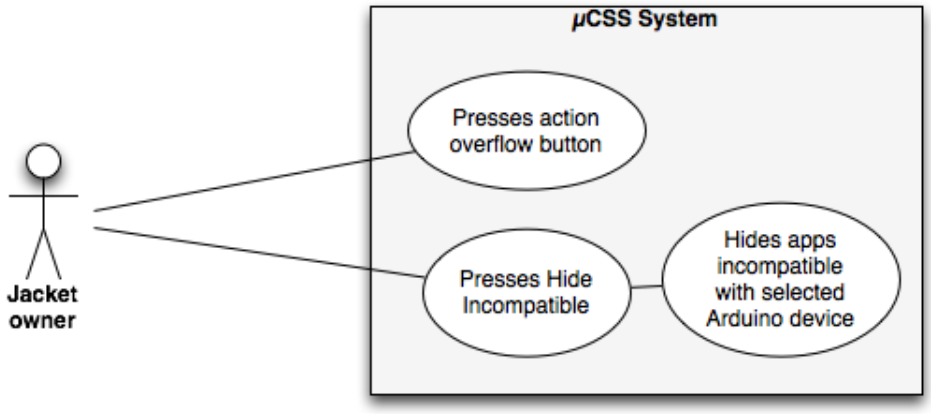


Figure 4.6: Use case 6: The user can open the action overflow bar and hide/show incompatible applications

The sixth use case describes how the user can choose to filter apps in μCSS based on their compatibility with the paired Arduino device.

Table 4.6: Use case 6

ID	6
Name	Hide incompatible apps
Actors	Arduino device owner
Prerequisite	Installed μCSS
Precondition	Actor has chosen a category
End Requirement	None
Main Flow	1. The user presses action overflow button 2. The user selects hide incompatible applications 3. The application hides all incompatible applications from the user 4. The user browses compatible apps
Alternative Flow	None
Parent UC	1
Child UC	None

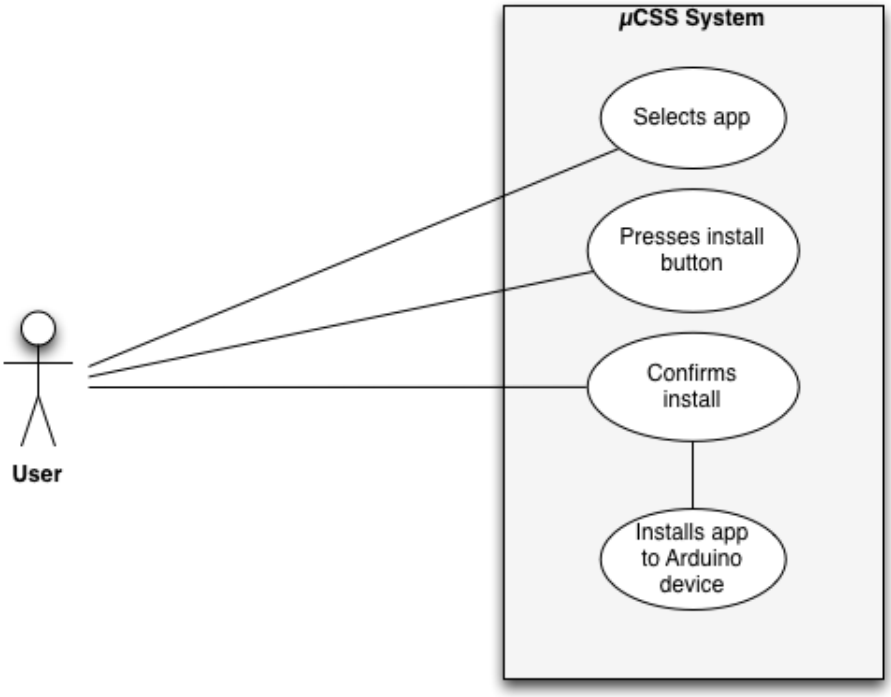


Figure 4.7: Use case 7: The user selects an application, presses the install button and confirms the installation.

The seventh use case describes the user sequence to choose and install an application from the μCSS to the paired Arduino device.

Table 4.7: Use case 7

ID	7
Name	Install application on Arduino device
Goal	Connect the Arduino device to the Android application with the use of QR Code
Actors	Arduino device owner
Prerequisite	Installed μCSS Arduino device connected to μCSS
Precondition	Actor on "Device list"
End requirement	The application is installed on the Arduino device
Main flow	1. The user selects the desired application 2. The user presses the "install" button 3. The user confirms the installation 4. The application is installed at the Arduino device
Alternative flow	3.a. The user cancels
Parent UC	1, 2, 3, 4, 5, 6
Child UC	None

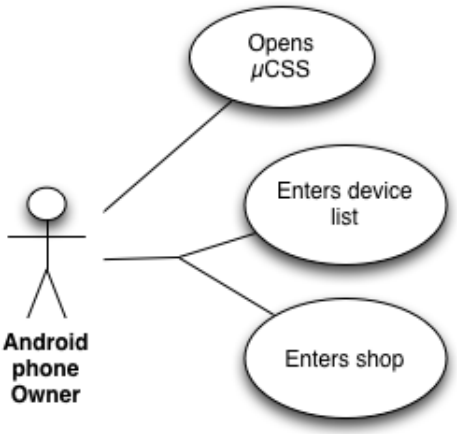


Figure 4.8: Use case 8: When the user starts the application this screen will be shown where the user can choose to select device list or browse the shop.

The eighth use case is the welcome screen which shows every time μCSS is opened.

Table 4.8: Use case 8

ID	8
Name	Welcome screen
Goal	Open the application
Actors	Android application owner
Prerequisite	Installed μCSS
Precondition	None
End requirement	The application is opened
Main flow	1. The user selects the μCSS application 2.μCSS is opened and the welcome screen is displayed
Alternative flow	None
Parent UC	1
Child UC	2, 3, 4, 5, 6, 7

5 | Development environment

In this chapter the different tools and resources used in relation with the project will be presented. A general requirement for a tool to be used, was that it was multi-platform, as three different operating systems were in use within the group. It was also preferable for the tool to be free, since this is a project that does not generate income.

5.1 Development tools

These are tools used in the development of the project. They have contributed to the technical aspects of development.

5.1.1 Integrated development environments (IDEs)

An IDE is used by programmers to develop and program applications. An IDE typically include a source code editor, build automation tools and a debugger. These are the IDEs that were used during the project.

5.1.1.1 Eclipse

Eclipse is a free IDE implemented in Java. It has extensive plugin support and anyone can publish a plugin to support another programming language, versioning software or new program features altogether. Eclipse provides useful, time saving functionality to developers, such as an extensive live debugging suite in addition to checking code syntax and providing auto completion of method calls.

Eclipse was chosen for Android development, due to previous experience with the software and good plugin support for targeting different versions of the Android API. Using a plugin for Arduino support was also considered, but the functionality of the available plugin was found lacking, and therefore not used.

Mylyn is a plugin meant to integrate with other plugins that provide access to task, issue and bug tracking repositories. Issues can be referenced or created quickly from the Eclipse IDE while coding, and the plugin can be configured to show relevant code sections when selecting an issue to work on. The group used a GitHub connector for Mylyn to get Mylyn to display and work with GitHub issues.

The Android software development kit (SDK) provides access to several versions of the Android API through the use of an installation manager. This allows for simple updating when new versions of the API are released, as well as supporting older devices. A simulator for testing against different Android devices is also available.

5.1.1.2 Arduino IDE

The Arduino IDE is provided by the creators of the Arduino platform as a free and open source program. It provides syntax highlighting, example programs, as well as basic editing tools. The IDE handles compilation of Arduino code into C and C++ code the regular microcontroller tools can handle and can pass it on to an integrated uploading tool (AVRDUDE) to get it running on the Arduino.

The official Arduino IDE was considered for coding the Arduino component in, but due to its limited text and code related feature set (lacking features such as auto complete), it was not used for coding.

5.1.2 Codebase management and versioning

The tools described in this section were used for versioning and managing the codebase.

5.1.2.1 Git version control software

Git is a free, open source Version Control Software (VCS) with support for both local and remote repositories. It keeps track of differences between versions of files and allows for offline commits and branch creation, as this is done locally first before pushing the updates to a repository.

There are various terminal and front-end solutions available, including a GUI version from GitHub and Eclipse plugins. Past experiences with Git plugins for Eclipse led to the decision to use terminal software, as the GitHub program lacked advanced functionality regarding branching [21], among others. Another benefit with that solution was that the VCS then was IDE- and platform-independent and that there was no need to use the IDE for working with the report repository.

5.1.2.2 GitHub

GitHub hosts repositories for use with the Git VCS. They also provide basic issue tracking and social features, such as following other developers or projects. Paying customers can elect to hide their code, while free users have to share their code with everyone (one user can however keep one hidden project as long as he is the sole developer). The customer required source code to be uploaded to GitHub and also recommended use of GitHub for requesting assistance using issues - a recommendation that was followed.

5.2 Project management tools

The tools mentioned here were tools used in general management of the project. This includes elements such as the planning of meetings, writing, and sharing information between group members.

5.2.1 Google Docs

Google Docs is a free to use online office suite that allows users to simultaneously edit documents of different types (text, spreadsheet, presentation, etc.), which was of great use

when the group was working together on documents. It also supports document chat and comments (temporary chat/forum thread-like structure).

5.2.2 Microsoft Word

Microsoft Word is the de facto standard Word Processor, with support for many different effects. Content creation and formatting tasks are nearly completely intertwined, making it tricky to maintain a consistent document as the documents grow larger. Microsoft Word was used to generate certain graphs for the documentation.

5.2.3 LaTeX

LaTeX is a macro-improved version of Tex used for typesetting documents. It is free and open source. The general idea is to separate content creation and formatting to create consistent documents effectively without being distracted by the appearance of the content.

5.2.4 Google Calendar

Google Calendar is a free to use online calendar. It supports both private and public events. Google Calendar allows for simple calendar synchronization across different devices and platforms. A shared calendar in Google Calendar was created to make it easier to keep track of group meetings, meetings with the customer, and the supervisor.

5.2.5 Dropbox

Dropbox is a cross platform file synchronization and sharing service hosted in the cloud. Files in a specified folder are automatically kept in sync across different devices. It is also possible to share folders with other users and with the world at large. The basic service is free, but comes with limited storage space. Dropbox was used for sharing various files within the group, such as compiled versions of the report, data sheets for Arduino components and so on.

5.2.6 TextMate and Sublime Text 2

These tools are sophisticated text editors that can be used for coding, markup and regular text editing. In the project they were used for editing of the report and coding for Arduino. While TextMate is for OS X only, Sublime Text is cross platform.

5.2.7 Web sequence diagrams

Web sequence diagrams is an online tool for generating sequence diagrams based on text markup. It provides both a paid solution and a free offering with a limited feature set [22].

5.2.8 Wunderlist

Wunderlist is a free to-do list tool supporting shared lists. It supports sharing tasks between the participants of a to-do list with reminders and notes. Wunderlist was used to serve reminders for other, non-coding related tasks, like scheduling meetings or booking rooms.

5.2.9 Doodle

Doodle is a free online tool for scheduling events. It allows a user to create an event and aims to simplify scheduling of events by “polling" the participants when they are available. When the participants of the event have answered, one can easily view when all participants are available for meeting. Doodle was used to schedule the first meetings the group had, before regular meetings were established.

5.2.10 Fritzing

Fritzing is a tool that lets the user draw circuit boards and wiring diagrams. This makes simple diagrams that are easy to understand and easy to reproduce for users that do not understand electronics. This tool is often used with the Arduino platform, because usually the users of this platform have no experience with electronics. Most of the standard electronic equipment used with the Arduino are included in this tool. Fritzing was used to create the wiring diagram of the Bluetooth wiring and the iJacket clone.

5.2.11 OmniGraffle

OmniGraffle is a software used to draw diagrams and charts. This program makes it easy to draw tidy diagrams and charts. OmniGraffle was used to create the ER-diagram and use cases for the documentation.

5.3 Test management

The tools mentioned here were used for preparation and creation of tests for the project.

5.3.1 Robotium

Robotium is a JUnit based framework for the creation of automatic unit testing in Android. It automates the created tests by going through the Android application as described in the tests. This shows the tester each step and simulated click on the Android application, allowing the tester to see if a mishap is due to the test itself being wrong or an error in the code.

6 | Design and implementation

In this chapter the design and system architecture of the product will be presented. The reader will learn how the different components of the product are put together, and how the design of the application was planned.

6.1 System architecture

The design was divided into several modules:

- Bluetooth connection between the Android and the Arduino
- Synchronization with the local SQLite database
- Android application view (General user interface)
- A service that contains the protocol for installing over-the-air

The system design was implemented such that further development should be as modular and easy as possible. Therefore it was designed as a plugin-like system where you easily can implement your own protocols against a desired device, e.g. Raspberry Pi. Currently the application only supports the STK500v1 protocol and therefore only connections with Arduino devices. With some work, compatibility with other STK500-based devices can implemented. be The design for the connection to other devices was done as shown in Figure 6.1.

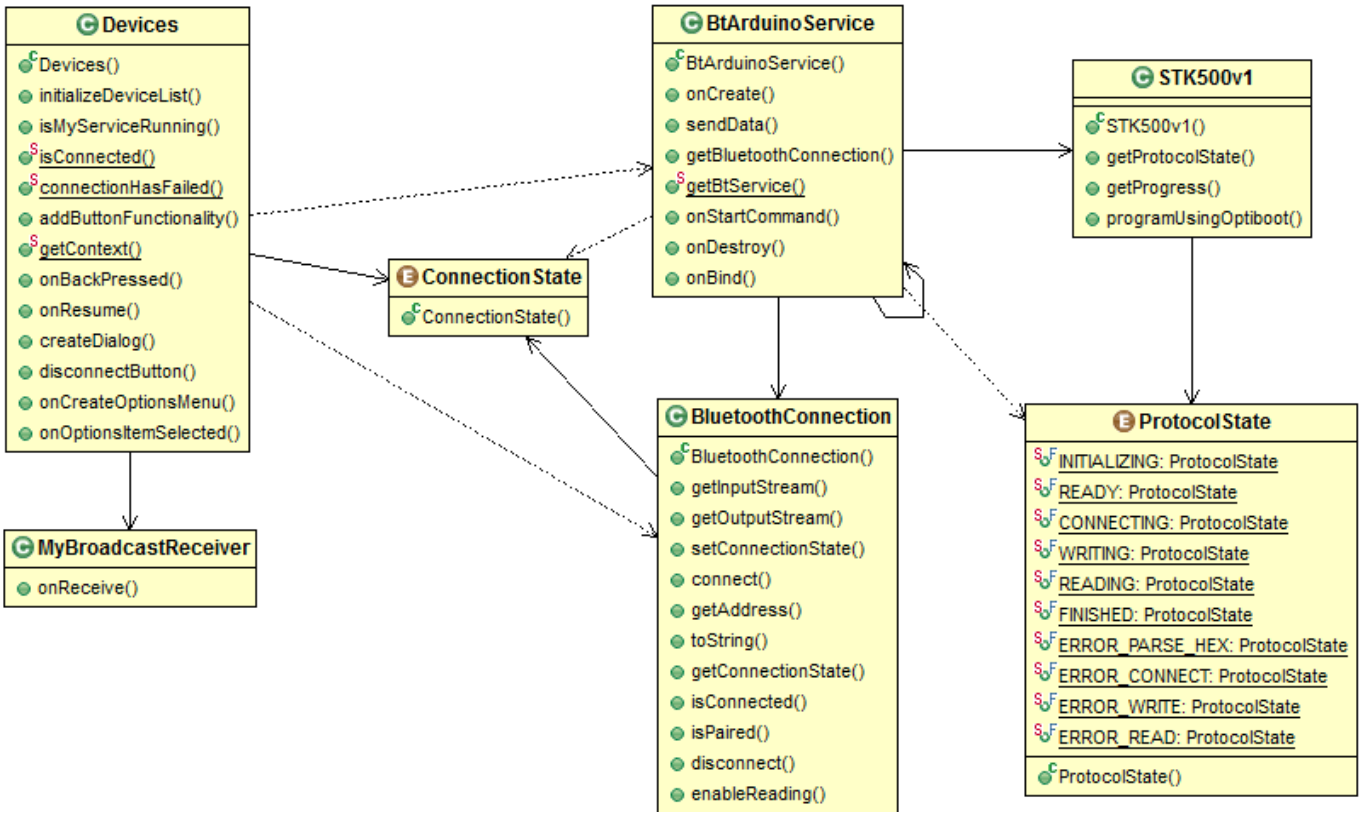


Figure 6.1: The Bluetooth connection is stored and managed in a service that provides the STK500 protocol.

Devices, as shown in Figure 6.1, is an activity that manages the discovered Bluetooth devices. A device is discovered by listening to the Bluetooth API on the Android. This happens in the *MyBroadcastReceiver* which works like a listener. When this listener gets notified with a device as input, the discovered device will be put in a list in *Devices*.

If the user choose to connect to a device from the list of discovered devices, a service (*BtArduinoService*) will be created. This service will create the *BluetoothConnection* that manages the connection between the Android device and a Bluetooth device. The service uses *STK500v1* for transferring data over the *BluetoothConnection* to the connected device.

To add support for other microcontrollers than Arduino, only few changes to *Devices* would have to be made.Another service and a protocol communicating with a desired device must be implemented. If the added protocol(s) should coexist with STK500v1, support for multiple simultaneous services must also be implemented as well. Since this project only considered connections between Android and Arduino, only the STK500v1 protocol and one service was implemented.

The overall design solution for multiple connections is described in Figure 6.2.

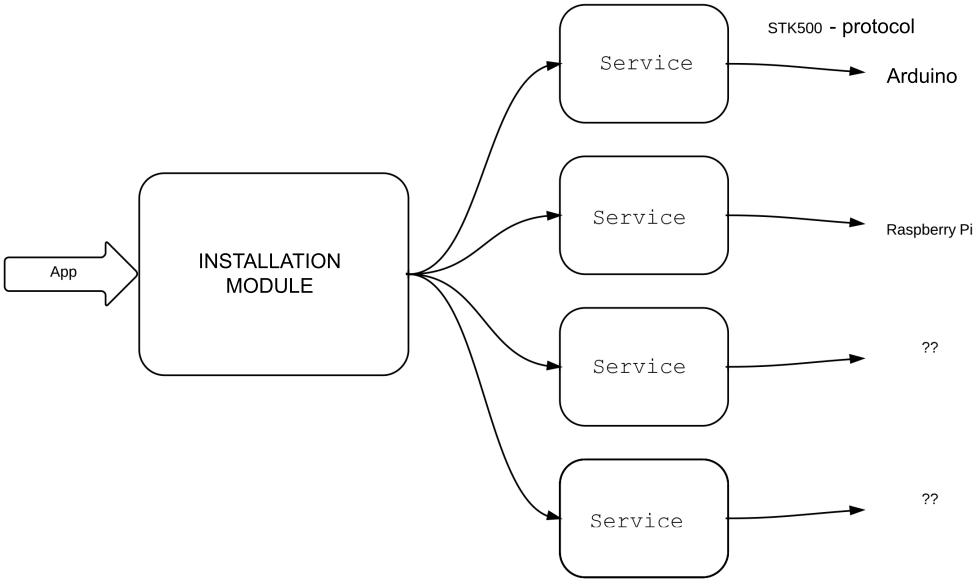


Figure 6.2: The installation module is a class that manages all the services that the application may support. Each service has a protocol for installing software over-the-air to a desired microcontroller.

The overall system architecture as shown in Figure 6.3 and Figure 6.4 illustrates the core components of the system and the relationship between these components. The *Market Application* is the centre of the application and is where most of the GUI and functionality visible to the user is shown. *Device List* is the list of devices that the user can choose from and connect to. This is also the location of the *Add Device* functionality, where the user can connect to a device in an alternative way (QR Code or MAC-address).

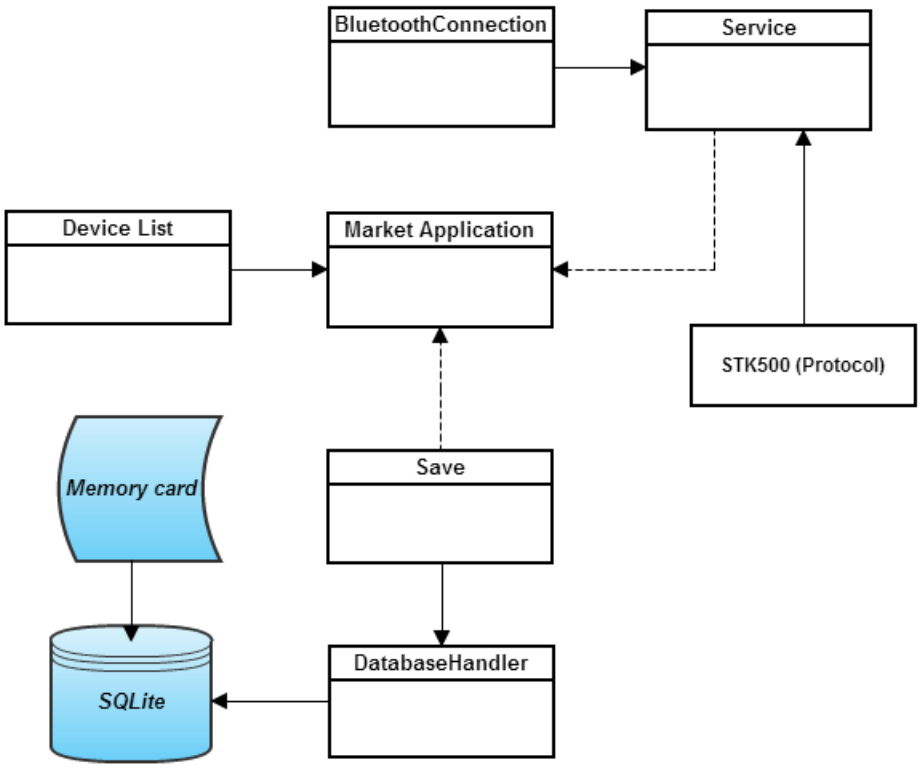


Figure 6.3: This shows the overall system architecture with the most important components.

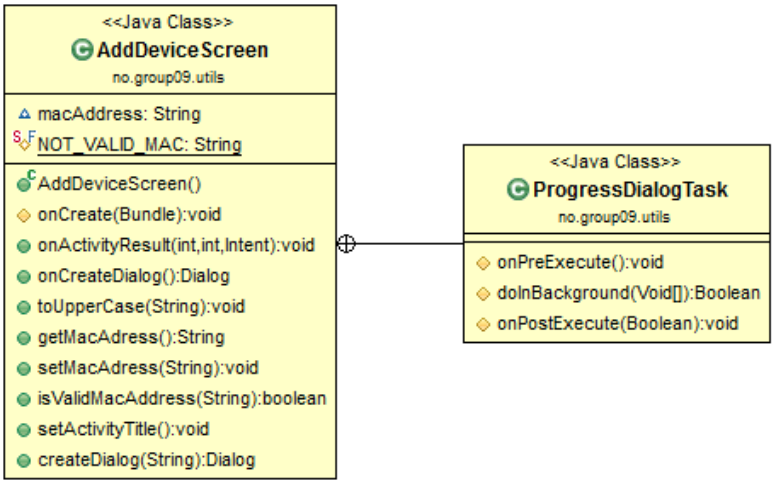


Figure 6.4: This is a UML diagram of the *Add Device Screen*. When QR-reader or input serial is chosen, a new asynchronous task is started that handles the Bluetooth connection process.

The *Service* holds and manages the Bluetooth connection (*BluetoothConnection*) and is communicating with the protocol for installing apps on the microcontroller.

The *Memory card* represents the local storage solution on the mobile phone where the database is stored. SQLite was used as the database engine because it is common and

easy to implement on Android.

DatabaseHandler is a class for taking care of the SQL transactions and is the access point for communication with the database. A helper class called *Save* is used for easy access to the *DatabaseHandler*. When the application communicates with the *Save* module, the database communicates with the DatabaseHandler. The *Save* class can be created from any class in the application. It ensures that only a single instance of the *DatabaseHandler* can be instantiated and handles insertions and queries.
Save was created because it can be initialized and created everywhere in the application (in optional Java classes) and makes sure it is only one instance of the *DatabaseHandler*.

6.1.1 Browse shop

This is the shop where the user can see the μC apps, choose categories and swipe through different fragments. In Figure 6.5 one can see the *MainActivity* which is the activity for the category selection of the shop. When a category is selected, a new activity *MainFragmentActivity* (see Figure 6.6) is started with the Intent about which category was selected.

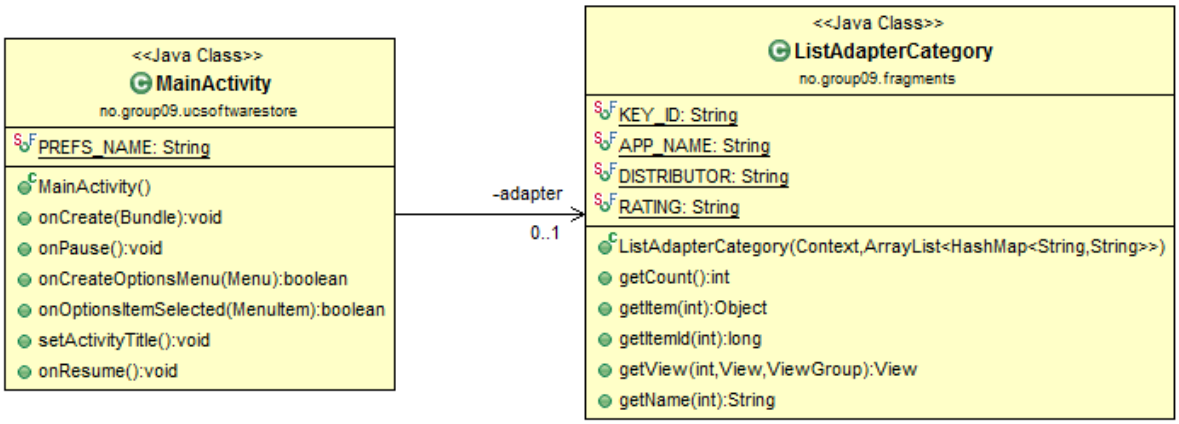


Figure 6.5: The *MainActivity* is an activity with a *ListView* that uses *ListAdapterCategory* as an adapter. When a category is selected, the *MainFragmentActivity* is started with Intent about which category was selected.

In Figure 6.6 the *MainFragmentActivity* is started when a category is selected. This activity creates a *FragmentPagerAdapter* that manages the fragments. One fragment is one of the views that can be swiped through. *All* and *TopHits* are these fragments. Each fragment consist of a list, and have an *Adapter* for this list (*ListAdapter*).

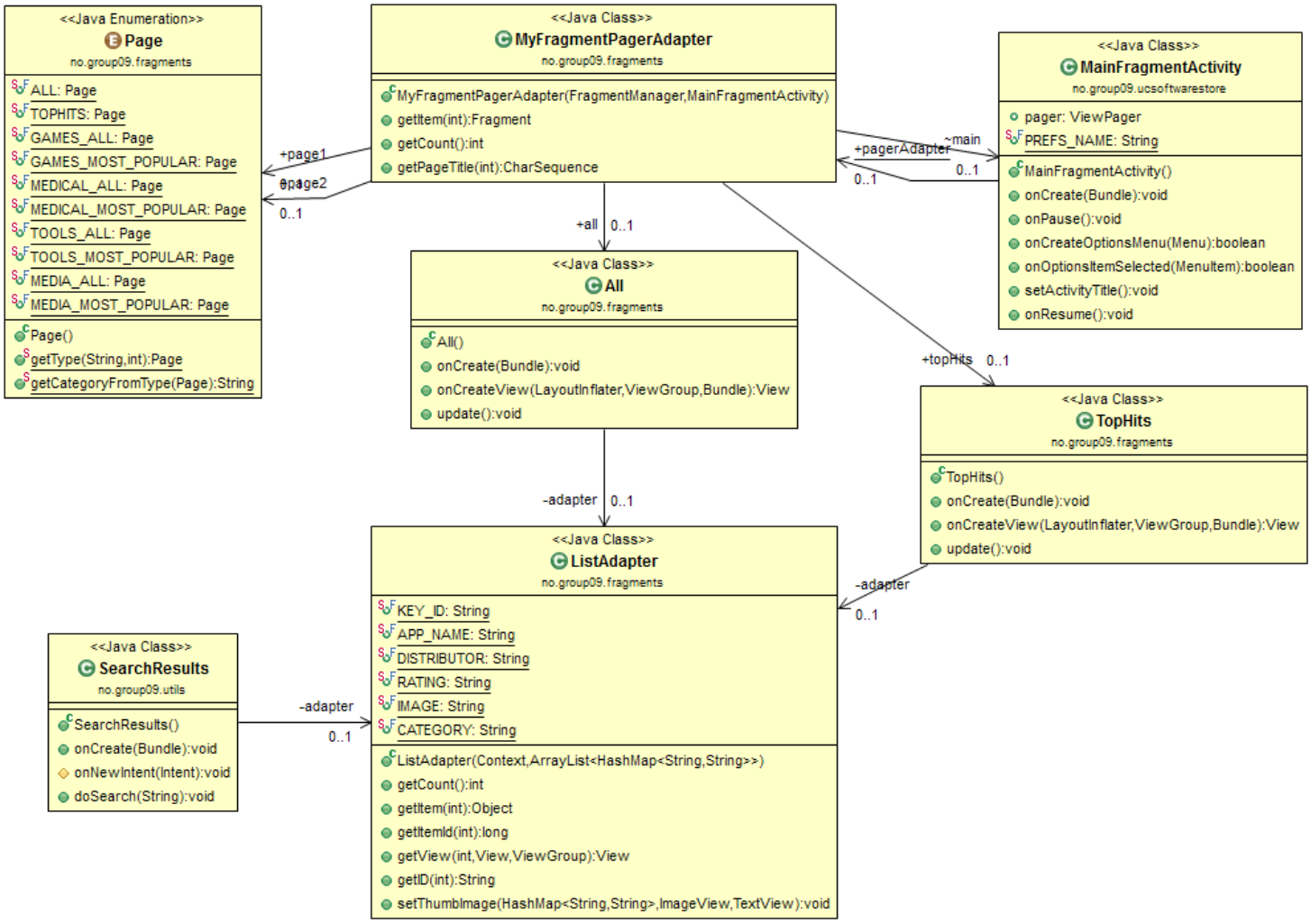


Figure 6.6: Here one can see a *FragmentPagerAdapter* that manages all the fragments. Each fragment uses a *ListAdapter* that structures the list items.

The *Page* enum is used to tell *MyFragmentPagerAdapter* which category that should be shown in *All* and *TopHits* fragments. This enum is sent as an intent to the *MainFragmentActivity* when started. When a category is chosen, the content in *All* and *TopHits* are filtered by only showing applications that belongs to the selected category.

SearchResults is a class that contains a list of search results (μC applications) from a search. This class is an activity that has a list like the other fragments, and therefore uses the same list adapter as *All* and *TopHits*.
When a specific μC app is chosen, a new activity is started: *AppView*. This class contains the information about the chosen app and presents it to the user. From this screen the user can choose to install the app on the connected Arduino.

6.1.2 STK500v1

In Figure 6.7 the general design of the protocol component is shown. The *STK500v1* class handles most of the logic and actual programming; this is the class called upon by the application when an app is to be uploaded. The *Constants* class holds important commands and responses used when communicating with the Arduino. The *Hex* class

deals with interpreting and verifying the app to be programmed. It provides the raw bytes requested by the protocol class.

The rest are for abstracting and dealing with reading from the Bluetooth connection; the current implementation allows for avoiding blocking by only reading buffered bytes. The Reader behaves differently while reading or during timeouts, based on a state pattern implementation. As shown, the different states are both states and readers.

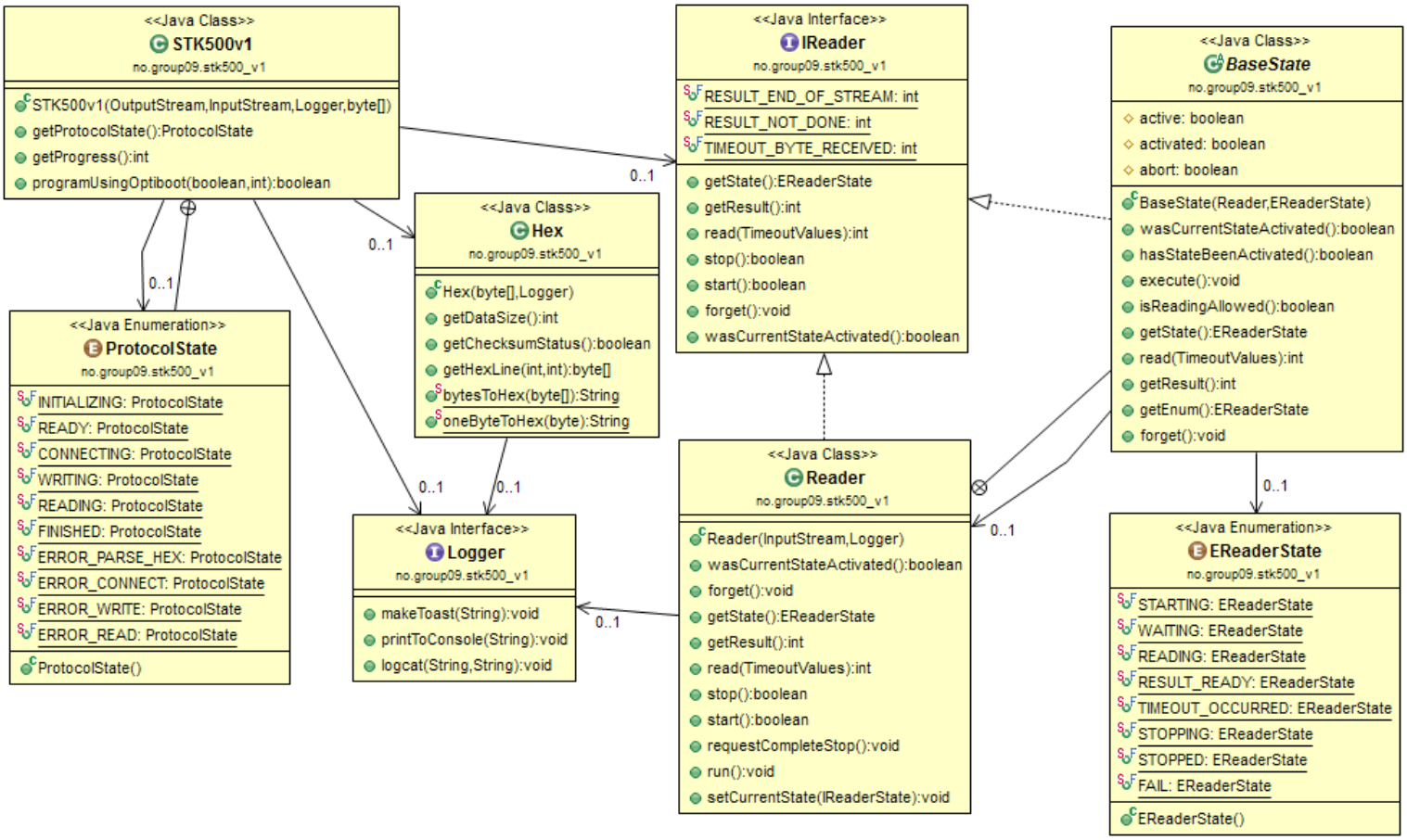


Figure 6.7: The most important classes used for programming

In Figure 6.8 a simplified view of the programming process is shown. Here the *ArduinoStore* is a representation of any application requesting an app to be uploaded to the Arduino. The *Arduino* represents a microcontroller to be programmed.

Note that the *IReader* has already been instantiated before the starting call, and that stopping doesn't destroy the object.

Most of the simplification occurs within the programming loop, as several different types of commands are sent to the device, depending on the responses received; for details on this communication, see the AVR061 documentation [16]. Communication details with the *Hex* class and *IReader*-related classes have also been reduced slightly in the diagram.

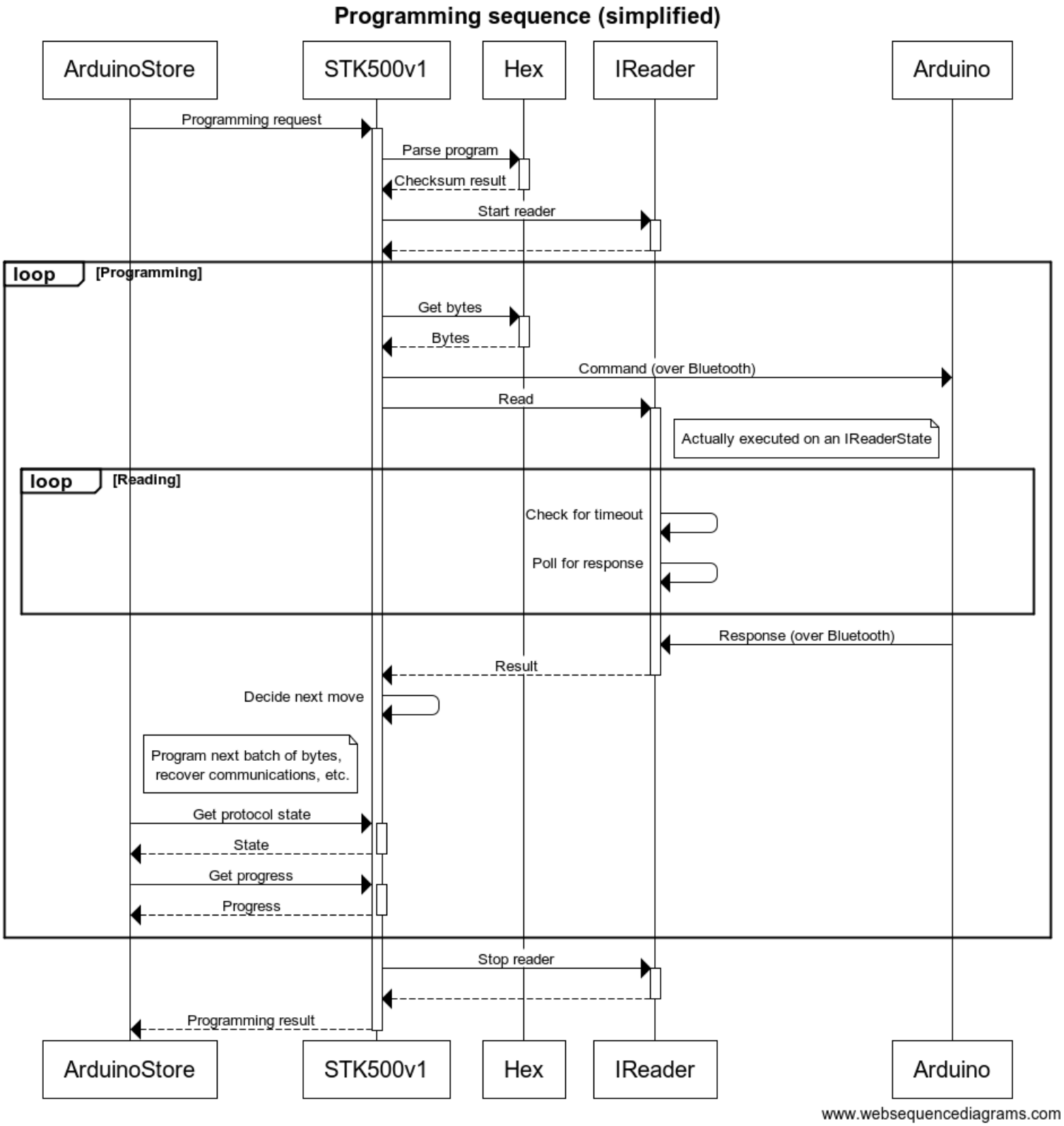


Figure 6.8: Sequence diagram of the programming process. The three classes in the middle are in the protocol library, *ArduinoStore* represents an outside caller, and the *Arduino* is a programmable microcontroller.

6.2 Design of the Android application

One of the purposes of the Android application was to ease the process of installing PUIs on an Arduino for non-technical users. As described in section 4.2, one of the requirements for the project was that people of all ages were supposed to understand how to use the application. This requirement put pressure on the design, as one has to make sure everyone were able to understand the different functions of the application.

6.2.1 Design guide

Before the programming of the Android application was started, a complete design guide was created. In this section the complete design of the application is presented. This guide was made primarily for two reasons:

- Presentation for the customer: With a complete design guide it was possible to present the user interface of the application to the customer before it was programmed. This allowed for input from the customer at an early stage, when it was easier to change the design.
- Avoid confusion: A design guide reduces the amount of confusion and discussion regarding the appearance of the user interface. Because the design of the user interface was settled on before the programming had started, there was less need to discuss this along the way.

During the project, some changes were made to the design. These changes can be viewed in Section 6.2.3.

Screen 1a - Device list, as shown in Figure 6.9, shows the list of available Arduino devices. In the final design the list of devices fills the whole screen, and the buttons and description text have switched places. When a device is pressed, a progress bar appears and stays on the screen until a valid Bluetooth connection with the chosen device is made.

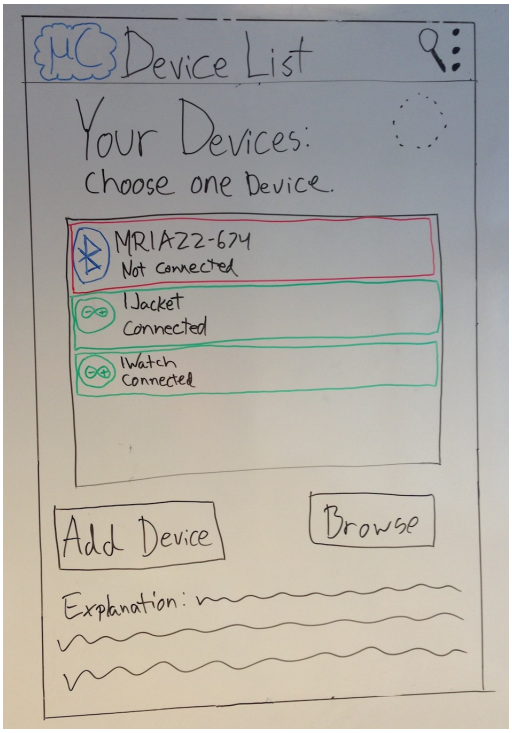


Figure 6.9: The design for the device list of the discovered Bluetooth devices

Screen 1b - Add device manually, as shown in Figure 6.10, appears after pressing the "Add device" button in Screen 1a. It was chosen to remove the "Bluetooth settings" button, as it proved unnecessary. In the final design, this screen contains only the "QR Code" button and "Input serial" button, with a short description of the functionality of the button between them.

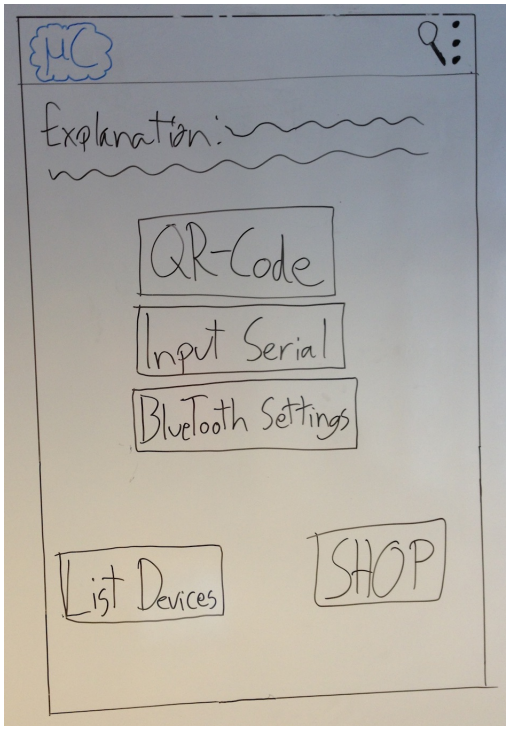


Figure 6.10: The design for alternative methods to add devices

Screen 1b-i - Input serial, as shown in Figure 6.11, appears when the "Input serial" button is clicked.

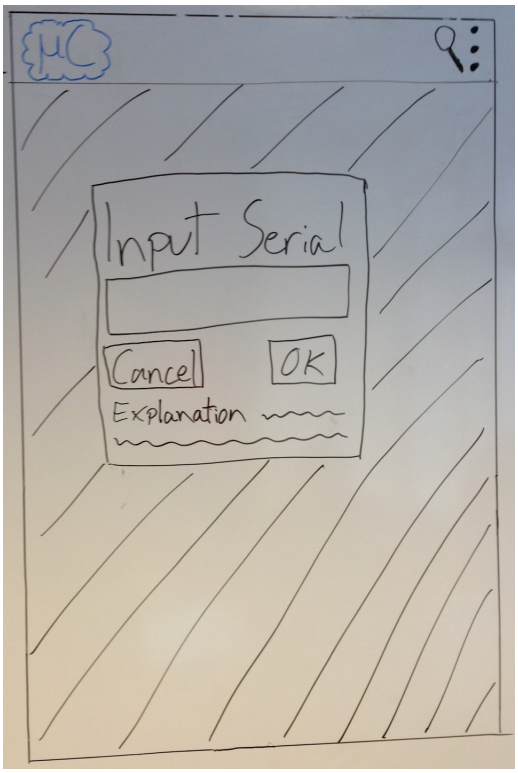


Figure 6.11: The design for Input serial box.

Screen 2a - Browse shop, as shown in Figure 6.12, is a screen for browsing all the apps for Arduino in the shop. More categories have been added. The user can swipe to the left or right to sort the available applications in different ways. See next paragraph.

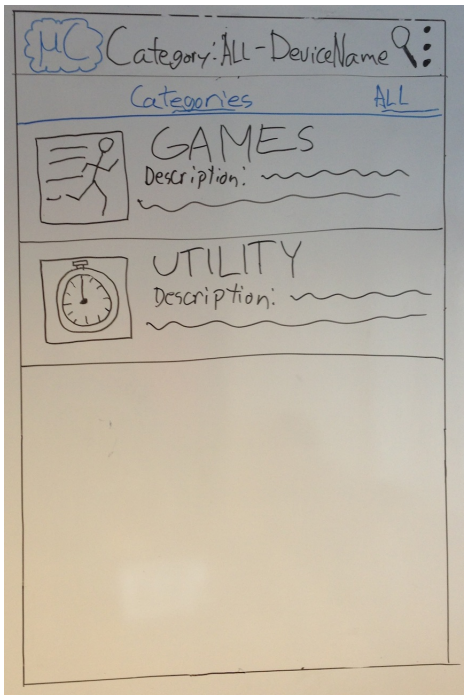


Figure 6.12: Design for category selection in the shop.

Screen 2b - Browse shop by category, as shown in Figure 6.13, is a screen that shows a list of all applications in the chosen category. Category "All" is chosen.

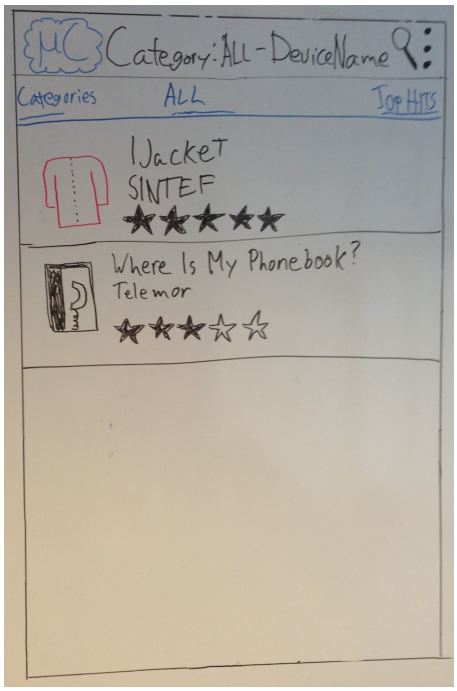


Figure 6.13: Design for the list of application that belongs to the selected category.

Screen 3a - Application view, as shown in Figure 6.14, provides an overview of a chosen application. Small changes were needed, as the comments field and reviews were given a lower priority at mid-term.

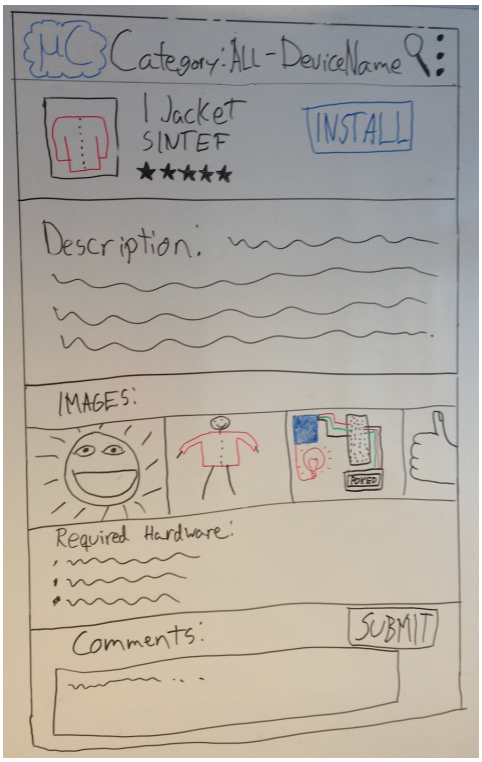


Figure 6.14: The design for application view. This is the screen you see when you look at the details about an application

Screen 3a-i - Installation confirmation, as shown in Figure 6.15, is a screen that appears when the "Install" button in Screen 3a is pressed. Shows the name of the chosen device.

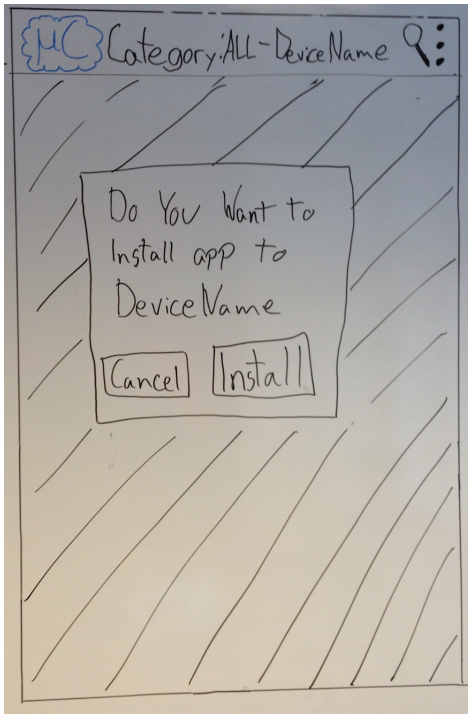


Figure 6.15: When the user click "install" , this box will show up to warn and confirm the users choice

Screen 3a-ii - Progress of installation, as shown in Figure 6.16, is a screen that shows the progress of the installation. The progress bar cannot be dismissed, so the Android application is locked until the installation is complete or has failed. The user is warned not to move the Android device out of range of the Arduino.

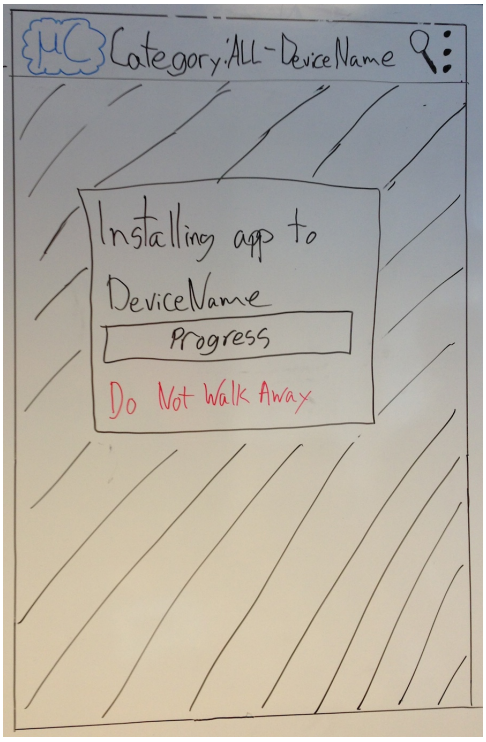


Figure 6.16: The progressbar that will shop up during the installation over-the-air.

Screen Xa - Action overflow, as shown in Figure 6.17, is a screen that appears when the "Action overflow" button is clicked. This menu is available from all the screen in the application with the exception of the preferences screen.

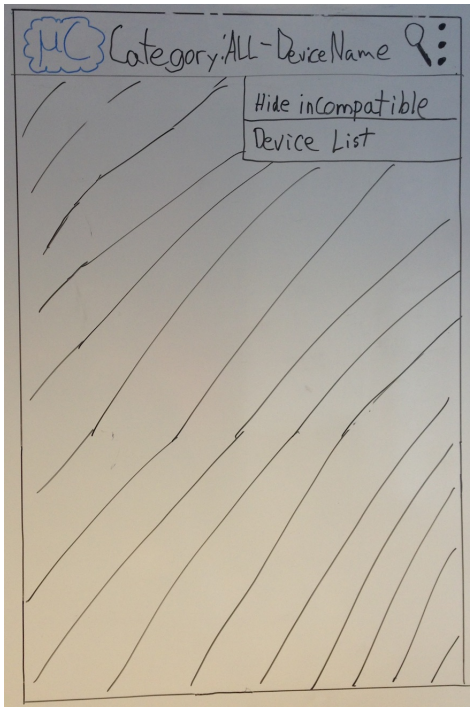


Figure 6.17: The menu bar that contains settings and properties.

6.2.2 Design decisions

This section explains the reasoning behind the different aspects of the design.

Device list. It should be simple and fit as many devices as possible, even on the smaller cell phones. As shown in Figure 6.9 it was decided to use discovered Bluetooth devices as a primary method of connecting to an Arduino. The less frequently used methods for connecting to an Arduino, like QR codes and serial input, were put in another screen, as shown in Figure 6.10.

Because of license compatibility issues, it was necessary to use external QR Code readers. To give the user more flexibility, it is therefore possible for the user to select the optimal QR-reader from his application list.

When *Serial input* is clicked, it was decided to open a dialog box for the input string as shown in Figure 6.11. This is because it clearly gives the user feedback on what is happening and it does not add unnecessary clutter to the GUI.

Browse shop. The category selection, as shown in Figure 6.12 groups the apps in different categories, making it easier for the user to browse for apps.

When a category is chosen, a new screen containing the apps in that category is created. In this screen is it possible for the user to swipe left and right to change the way the apps is sorted. This swiping was implemented because it easily and quickly selects what to show. "Top hits" and "All" were chosen because the database is currently small, and it is not necessary with a large amount of "sort by" tabs. Other tabs could be easily implemented if need be.

Application view. The application view, as shown in Figure 6.14, should show all the details and information about an application that is useful and interesting to the user. Rating, description, developer and application name were selected for trying to keep a "standard". Meaning that both Google Market and AppStore uses these elements and are expected to be familiar to most Android users.

When the "Install" button is clicked, a confirmation box will pop up as shown in Figure 6.15. This is because the user might have clicked the button by accident, or is not informed about what device he or she is connected to. The device name will therefore be shown in this dialog box.

A progress bar will pop up when the user have confirmed the installation. This is because the application should give an indicator on when the application is complete and when its safe to close the application.

Action overflow menu. When the user clicks on the action overflow menu, he will expect some sort of settings. That is why this menu contains settings. If the user will hide or show incompatible application to his Arduino, this menu will toggle the preference as shown in Figure 6.17.

6.2.3 Changes to the design

In this section the changes made in the design guide will be presented. The final design of the application can be viewed on an Android device.

Browse shop. In the "Browse shop" screen, shown in Figure 6.12, the swiping function was removed due to usability issues. Peer reviews indicated that it would be more user friendly to have this as a separate screen, instead of a fragment as shown in the design guide.

Application view. Due to lack of time, some simplifications were made to this screen. It was decided to remove the images and functionality for adding comments was not implemented.

Action overflow menu. Several different options were added to the Action overflow menu. It is convenient to show different options in this menu, depending on which screen the user is on, so the content of the menu changes throughout the application.

Add device manually. The "Bluetooth" button in Figure 6.10 was removed, as it was unnecessary.

Welcome screen. The "Welcome Screen" was added as the start screen. This makes it easier to navigate between "Devices" and "Browse Shop". It also provides an option (checkbox) that allows the user to choose whether or not they will automatically reconnect the next time the application is started. Figure 6.18 shows a simple connection between the reconnect option and the welcome screen.

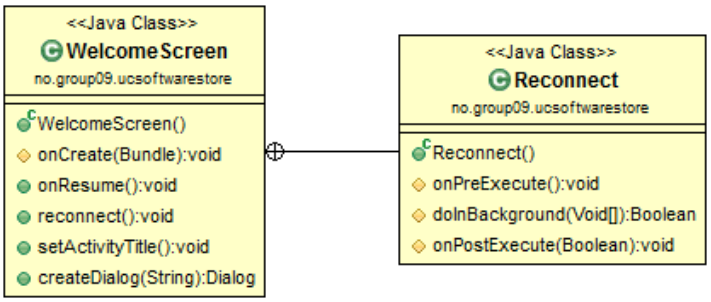


Figure 6.18: The Welcome Screen will reconnect to last connected device if the user have this option enabled. When the user clicks on "Devices" or "Browse Shop" a new activity is started.

6.3 Database implementation

SQLite was used as database language because it is integrated with Android and have much functionality that makes it easy to use in an Android application. Because of changes to priorities, the group and the customer agreed to drop the sync adapter, and only use a local database.

6.3.1 Database model

In Figure 6.19 the database is shown as an ER Diagram. Established appstore features, like ratings and application pictures, were not important to the customer. Because of this, the *pictures* table was not used.

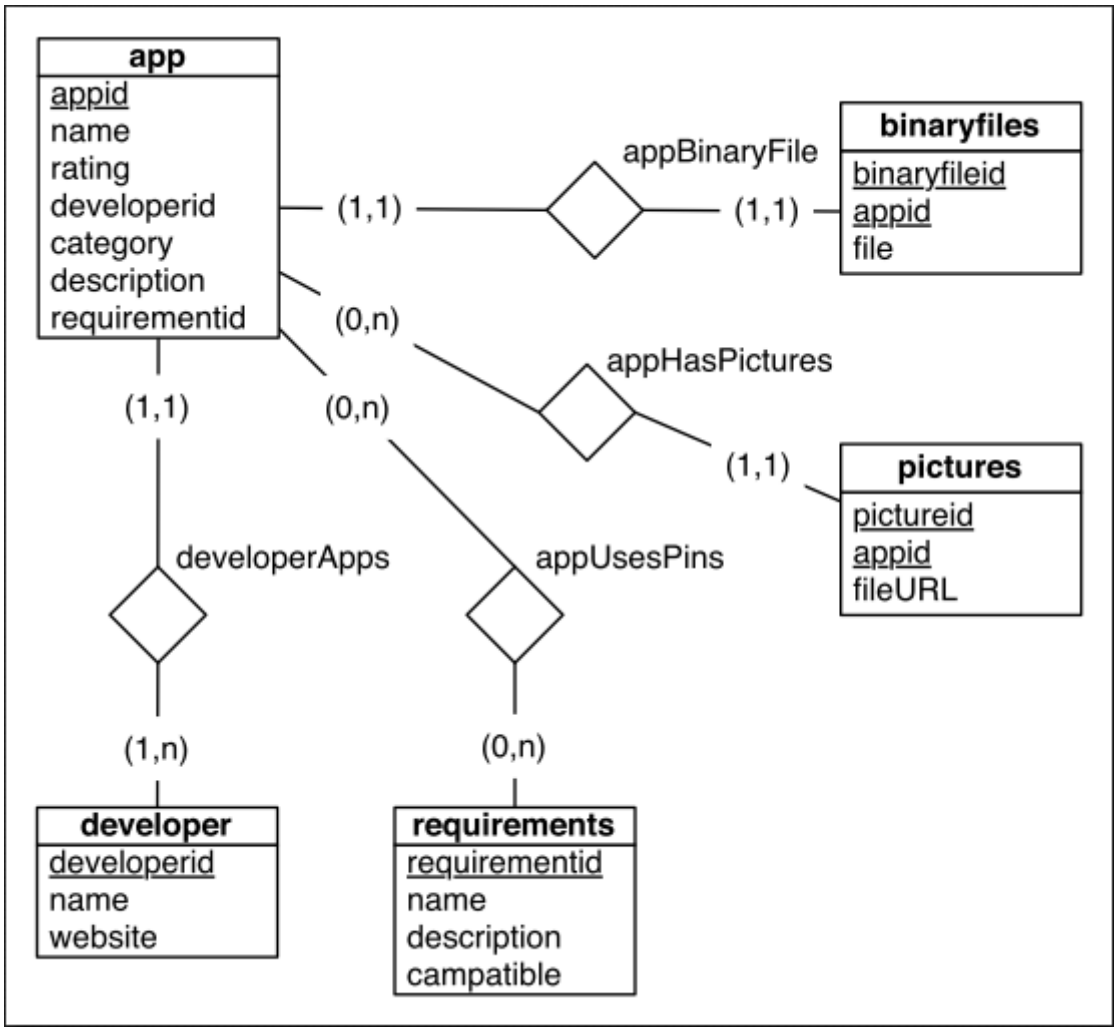


Figure 6.19: The ER Diagram describing the setup of the database.

6.3.2 Database details

This section describes the structure and details of the database tables.

6.3.2.1 Table: app

This table contains information about the application.

appid(integer): auto-increment primary key for applications
name(varchar 160): name of the application
rating(int 10): rating of the application
developerid(int 10): foreign key to the developer of this application
category(varchar 200): which category this application belongs to
description(varchar 200): description of the application

requirementid(int 10): foreign key to the requirements of this application

6.3.2.2 Table: binaryfiles

This table contains information about the hex-file (compiled Arduino code).

binaryfileid(integer): auto-increment primary key for binary files
appid(int 10): foreign key to the application that owns this binary file
file(BLOB 1000000): The binary file in a BLOB

A BLOB is not a datatype and stores data exactly as it was input. This database stores it as a binary array.

6.3.2.3 Table: developer

This table contains information about the developer.

developerid(integer): auto-increment primary key for developers
name(varchar 160): name of the developer
website(varchar 200): website of the developer

6.3.2.4 Table: requirements

This table contains the requirements for the application.

requirementid(integer): auto-increment primary key for the requirement
name(varchar 160): name of the requirement
description(varchar 200): description of the requirement
compatible(int 10): integer that is converted to boolean when retrieved from database.

Compatible is an integer of 1 or 0. This indicates if the application referred to is compatible with pagers or not.

6.3.2.5 Table: appusespins

This table contains the information about which pins on the Arduino that the application is using.

appid(int 10): foreign key to the application that uses this rule
requirementid(int 10): foreign key to the associated requirement

7 | Testing

This chapter contains a description of the testing performed during the project. Tests were performed in order to ensure product quality and that the product met the requirements. The first sections describe how the tests should be performed, followed by the execution and results of the tests.

7.1 Testing strategy

The ways in which testing was performed is detailed in this section. Unit testing was performed initially, followed by the integration testing. Following the completion of the Android application, usability testing was executed. User acceptance testing was then performed by the customer after the significant parts were completed [8].

7.1.1 Unit testing

For testing of the Android application, Robotium was used. Robotium is a powerful test framework that can be used for function, system and acceptance testing. The tests created in Robotium aims at writing small portions of code and checking the results against expected predefined values. Robotium was used for unit testing on the Android application. During the development of the application, different units were to be tested with Robotium after they were finished. An example of a Robotium test is shown below. In this test it is asserted that the Bluetooth connection with an Arduino device is as expected [9] [10] [11].

```
solo.clickOnActionBarItem(R.id.settings);

ConnectionState expectedState = BluetoothConnection
        .ConnectionState.STATE_CONNECTED;

ConnectionState actualState = BtArduinoService.getBtService()
        .getBluetoothConnection().getConnectionState();

assertEquals(expectedState, actualState);
```

7.1.2 Integration testing

Integration testing is based on combining different units and testing the interface and communication between them. This type of testing comes after the unit testing phase and before the validation testing [17]. The units that can be used in these tests must therefore have passed the unit tests. This made integration testing focus purely on the integration of units, rather than testing that the units were working correctly as

well. Because the system is composed of more than one component (STK500 protocol and the Android application), they were to be tested in pairs and not all at once. Integration testing identifies problems that occur when different units of the product are combined. These test were to be performed when units of the product were put together [18]. Just as in unit testing, Robotium was to be used for integration testing as well.

7.1.3 Usability testing

As the intent of the product was to ease the use and setup of physical user interfaces, usability tests were crucial in the testing aspects. The decision was made to perform a hallway test, where random test subjects followed the tasks detailed in Section 7.1.3.1. To ensure that the test subjects had a proper understanding of the product, the think aloud protocol was to be used, meaning the subject was supposed to be talking about their actions during the test.

All test subjects filled out a System Usability Scale (SUS) form in order to create a score, 0-100, which gave us a comparable score with other systems. Concerning the amount of subjects tested, five users has been recognized as the proper amount. According to Nielsen, 85% of the problems with a system will be found by these five users, thus testing more would increase the time spent watching the same problems resurface [12].

7.1.3.1 Tasks

These were the tasks given to each tester. The tasks were created with the use cases mentioned in Chapter 4.4 as a basis. Each task was meant to have a vague description. This way, test subjects would have to think for themselves, instead of simply following detailed steps.

The preface of these tasks was that the test subjects had bought a jacket with an Arduino device fitted in it. This jacket was called an iJacket and they had just downloaded the μCSS in order for them to check what other applications they could install on the device.

1. Select your jacket
2. Browse applications
3. Install the chosen application

After the tests, the test subjects were to be asked if at any point during the test they got stuck or confused at any task. The feedback from the test subjects were to be recorded.

7.1.4 User acceptance testing

User acceptance testing was to be performed with the customer in order to ensure that requirements were met and to check whether or not the customer was satisfied. This test was to be done in much the same way as the usability. A phone with the application was given to the customer in order for him to get first degree experience with the application. The tester was then asked to perform the same cases as in the usability tests, while a group member recorded his reactions to the application.

There were also expert reviews held during a presentation at the offices of SINTEF. The think aloud protocol was also used here. These testers had much experience within development, but not much knowledge about the task. They were therefore given a brief presentation beforehand about the project and given a short presentation of the application itself.

7.2 System testing

In this section the system tests will be presented. Each test was to be written to test a specific part of the product. This applies to both the Android application, STK500 protocol and software installed on the Arduino device. Each test was to be executed separately, and the results will be presented in Section 7.3.

Table 7.1: Connect with device using device list

Test ID	ST01
Test name	Device list connect
Test description	Connect with an Arduino device from the Device list screen
Precondition	Arduino device is switched on with Bluetooth enabled
Test steps	• Start program • Press "Device" list from Action overflow or from the Welcome screen • Press desired Arduino device
Success condition	Android and Arduino device are connected via Bluetooth

Table 7.2: Connect with device using QR Code

Test ID	ST02
Test name	QR Code connect
Test description	Connect to an Arduino device using QR Code reader
Precondition	Arduino device is switched on with Bluetooth enabled
Test steps	• Start program • Press "Device" list from Action overflow • Press "Add" device • Press "Connect with QR Code" • Choose QR Code reader • Scan QR Code
Success condition	Android and Arduino device are connected via Bluetooth enabled

Table 7.3: Connect with device using serial

Test ID	ST03
Test name	Input serial connect
Test description	Connect with an Arduino device using serial
Precondition	Arduino device is switched on with Bluetooth enabled
Test steps	• Start program • Press "Device" list from Action overflow • Press "Add" device • Press "Input serial" • Enter MAC address
Success condition	Android and Arduino device are connected via Bluetooth

Table 7.4: Search for desired app

Test ID	ST04
Test name	Search
Test description	Search for a specific app
Precondition	Database is populated
Test steps	• Start program • Press search icon • Type search string
Success condition	Search result is shown with matching apps

Table 7.5: Connect to last connected device on application startup

Test ID	ST05
Test name	Last connected
Test description	Test that the Android application connects to the last connected Arduino device when the application is started, and gives proper feedback to the user. The application should only reconnect when this option is selected.
Precondition	Available Arduino device with Bluetooth
Test steps	• Check option "Automatically try to reconnect to last connected device" • Connect with an Arduino • Exit Android application completely • Start Android application – When the Arduino device is in range – When the Arduino device is out of range/off
Success condition	When the last connected Arduino device is on and in range, the Android device should connect to it immediately when the Android application is started and give feedback to the user. If the Arduino device is out of range or off, proper feedback should be given

Table 7.6: Browse apps test

Test ID	ST06
Test name	Browse apps
Test description	Browse apps in the Android application in a random fashion
Precondition	Database is populated
Test steps	• Start program • Choose a category • Choose an app • Go back and choose different app • Swipe left or right to browse different sorting
Success condition	User is able to browse apps

Table 7.7: Install application on Arduino device

Test ID	ST07
Test name	Install application on Arduino device
Test description	Choose a desired application to install on a connected Arduino device
Precondition	Android and Arduino device is connected via Bluetooth
Test steps	• Start program • Select desired app • Press "Install" • Press "Confirm"
Success condition	Arduino app is installed on the connected Arduino device

Table 7.8: Change connected device

Test ID	ST08
Test name	Change connected device
Test description	Test that it is possible to first connect to one Arduino, then another, thus changing which device the Android application is connected to
Precondition	Two available Arduinos with Bluetooth enabled
Test steps	• Start program • Connect with Arduino #1 • Connect with Arduino #2
Success condition	Both connections are successful, and the Android device is connected with Arduino #2

7.3 Test results

In this section the results of all the tests performed will be presented. In each subsection the result of the given type of test will be shown.

7.3.1 Unit testing

All written Robotium tests run without encountering errors. This indicates that the units tested work properly and correct values are obtained throughout the testing. A decision was made to not write any more tests, as time was of the essence and resources were needed elsewhere.

7.3.2 Integration testing

When different units of the product were ready to be sewn together, integration testing was performed. If bugs and errors were encountered, the person currently responsible for the integration testing would fix it.

7.3.3 System testing

The system testing was performed according to the description of the tests in section 7.2. In table 7.9 the results of each test is shown. These tests were performed when both unit testing and integration testing had been done to ensure that the product met the requirements set by the customer.

Table 7.9: System test results

Test ID	Result	Comment
ST01	Passed	If the Arduino device is on and within range, the Android application successfully connects to it via Bluetooth using the Device List.
ST02	Passed	If the Arduino device is on and within range, and the QR Code is correct, the Android application successfully connects with it via Bluetooth using optional QR reader.
ST03	Passed	If the Arduino device is on and within range, the Android application successfully connects with it via Bluetooth by providing the correct MAC address of the device to the Android application.
ST04	Passed	It is possible to perform searches for apps in the Android application. Apps that fit the search criteria is presented to the user.
ST05	Passed	The application automatically tries to reconnect to the last connected device when this option is selected. It gives proper feedback to the user containing the result of the connection attempt.
ST06	Passed	It is possible to browse apps in the Android application in a random fashion without problems.
ST07	Passed	It is possible to install an app on an Arduino device through Bluetooth.
ST08	Passed	It is possible to change the connected device without disconnecting first.

7.3.4 Usability testing

The results of the usability testing were positive overall. The scores varied from 57,5 to 90 out of a 100 with an average of 78,3. In the end, six people were tested with different technological backgrounds. Not many problems occurred during the testing. One person did, however, have a small complaint about the consistency of the design of the application. After discussing it, it became apparent that this issue came from this person not being used to Android devices.

7.3.5 Expert reviews

After the presentation by the customers request at the SINTEF offices, three people agreed to fill out SUS forms about their thoughts of the system. As there were still problems with the system, the scores reflected the criticism of expert uses, but received an average score of 54 out of a maximum of 100. The main ideas for improvements were stability and security.

Stability was then worked on, but the security aspect was not recognized as an important focus point this late in the development. This was agreed upon by the customer after the meeting.

7.3.6 User acceptance testing

A formal user acceptance test session was not held, but during the last meeting with the customer, he got the opportunity to test the product. The customer was pleased with the performance of the system, and gave positive feedback.

7.3.7 Summary

Based on the tests performed during and after development of the product, Table 7.11 describes to what degree the initial requirements set by the customer were met. As described in the Table 7.11, some of the requirements were omitted during the project as some tasks proved more time consuming than expected. As a result of this, not all the initial requirements have been met.

Regarding the non-functional requirements, it is difficult to summarize to what degree some of these requirements have been met. As a result of this not all the non-functional requirements are included in the table below.

Table 7.11: Functional requirements

Requirement code	Result	Comment
FR01	Passed	The Android application is communicating with the Arduino device, and it is possible to install all the apps featured in the Android application on the Arduino over-the-air.
FR02	Passed	Usability testing was performed at SINTEF workshop, and in the hallway at NTNU.
FR03	Passed	Five example PUIs were made to prove the functionality of the product
FR04	Passed	A valid Bluetooth connection stays active even if the Android application is minimized. When the application is killed and restarted, it will connect to the last connected Arduino device if in range
FR05	Invalid	In agreement with the customer, this requirement was omitted
FR06	Passed	The user of the Android application can connect with an Arduino using QR Code reader, manually input a MAC address or choose from a list of available Bluetooth devices
NFR02	Passed	The Android application is stable, and the Bluetooth connection is persistent throughout the program
NFR03	Passed	All code written is open source. Code belonging to the Android application is under the Apache 2.0 licence

Continued on next page

Continued from previous page

Requirement code	Result	Comment
NFR04	Passed	The Android application is compatible with at least Android 2.3 and newer

8 | Roadmap for further development

This chapter describes what work remains in this project and a pointer to how this could be done. The different points of improvement mentioned in this chapter provides an overview of what the group would have focused on if given more time.

8.1 The μC Software Store application

This section focuses on areas of improvement regarding the μC Software Store. Some of the points mentioned in this section are important to elaborate on for the application to be fully usable to the general user, while others are of less importance.

8.1.1 Database

One of the most pressing matters that needs to be elaborated in future work is the implementation of a remote database. The delivered version of the application uses a local database that is automatically filled with data when the application is installed on an Android device. The use of a local database makes it difficult for general users of the application to add their personal apps to the μC Software Store database. This is because these apps would need to be hard coded into the Android application. With more time available, the group would have implemented a remote database that would contain all the apps that would be offered through μC Software Store. Content providers and sync adapters provided by Ubibazaar could be used to achieve this.

8.1.1.1 Uploading interface

Although outside the scope of the assignment given for the project, future work should encompass implementing an interface for easy uploading of apps to the μC Software Store. A possible solution for this could be something like the system used by Google Play. Here, registered users easily can submit and upload their own application through a web interface provided by Google Play. Allowing users to upload their own apps would enrich the product greatly, opening for a much wider user group.

It is envisioned that the use of a separate web site for uploading of apps would be best for this solution. On this web site the users should be able to view the existing apps in the database as well as submitting their own; much like *http://play.google.com*.

Security. Allowing individual developers to upload their own apps to the database would require an administrator or moderator to review the submitted apps. This is important to

avoid uploading of malicious apps and duplicates in the system.

8.1.2 Standard for defining μC Software Store compatible microcontroller-based devices

As part of the assignment, the group was supposed to implement filtering of apps that is incompatible with the connected device. As mentioned in section 4.3.1 this requirement was omitted due to lack of time. Instead of implementing the actual filtering into the application, it was decided that the group should produce a standard for defining μC Software Store compatible microcontroller-based devices. This standard can be viewed in appendix C.

Future work should encompass implementing this, or a similar standard, into the μC Software Store. Allowing users to filter apps according to compatibility with the connected device will empower the application and increase its usability.

8.1.3 Performance

Given more time, the group would have had a higher focus on the performance of the μC Software Store. Although the performance is satisfactory, it is possible to enhance it. Future work on this issue should encompass profiling the application to identify non-optimized sections in the code. Optimization of battery usage should also be looked into.

8.1.4 GUI

The Android developer design guide teaches the developers of Android applications how to design applications that fits the Android platform. Further work on the μC Software Store should make sure that the application follows the design guidelines provided by the Android Developer Guide [20]. This implies correct use of colours, scaling of the application to fit different screen sizes, use of patterns etc.

8.2 STK500 and over-the-air installation

This section focuses on the areas of improvement regarding the implementation of the STK500 protocol, and over the air installation of apps on Arduino. Although the functional requirements set by the customer were met and it is possible to install apps on an Arduino using the μC Software Store, there are still some areas of improvement.

8.2.1 Hard reset

There was not enough time to fully implement neither the hardware- nor the software implementations of hard reset. The former due to soldering difficulties and lacking documentation, and the latter had to wait for the former (though sections in the code where the feature should be enabled was identified).

8.2.1.1 How to hard reset

Both Bluetooth devices experimented on (RN-42 [6] and HC-05 [7]) provide pins that can be used for detecting when it makes a connection. Passing the output to the reset pin of the microcontroller (coupled with a transistor to make it able to coexist with the soft reset) will

then cause it to reset, as shown in Figure 8.1. The group has confirmed that this will work.

Triggering a hard reset is then as simple as disconnecting from the Bluetooth device and then reconnecting again. Timing is important, however, as the protocol code must communicate with the boot loader before the latter attempts to start the installed app.

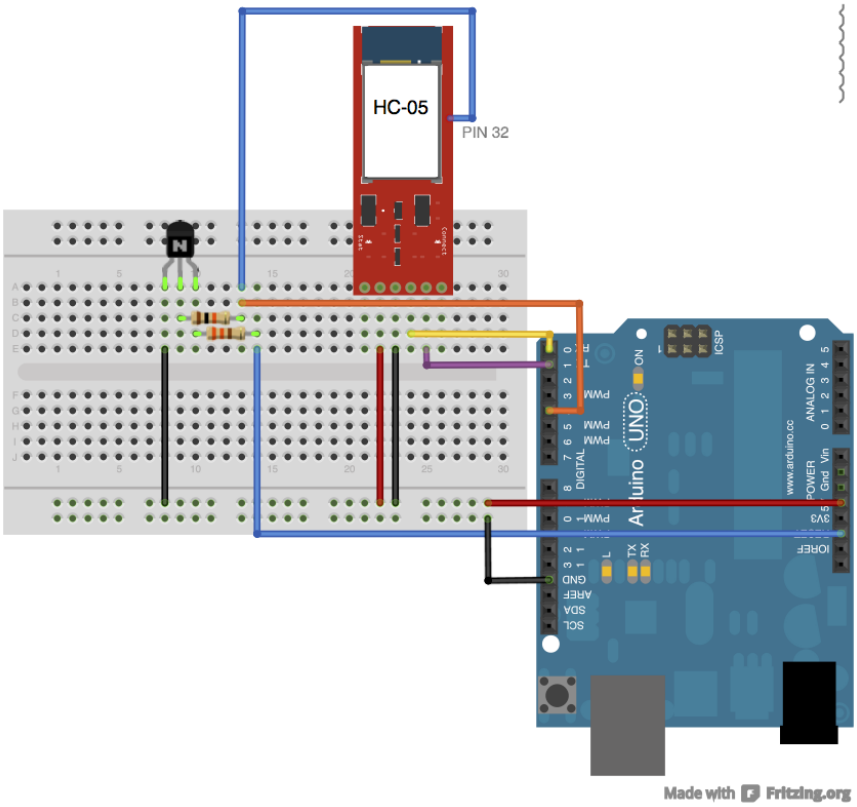


Figure 8.1: How to wire the hard reset using the HC-05 [7].

8.2.2 Security

Bluetooth itself provides security measures through channel encryption and pairing devices. In the case of critical devices, additional security might be needed. A simple way to incorporate this, would be to store a key in the EEP-ROM of the Arduino. The ComputerSerial library would then require authentication based on this key before allowing the initial soft reset. A QR code (this should not be plainly visible) could contain both the MAC address of the device and the required security details for ease of use, though a manual input method should also be available. This could easily be integrated with the regular connection management present in the application.

However, this implementation is vulnerable to hard resets; if the device restarts on a Bluetooth connection, the security scheme is pointless. Because of this, the hard reset feature would have to be disabled on such devices, meaning the added security is at the expense of ease of use. This presents a second problem; without a way to be sure of automatic error recovery, even authenticated programming poses a risk to critical devices.

Therefore, where more than paired device security is required, regular cabled programming is the safest way to go.

8.2.3 Protocols

The implementation targeted specifically the Optiboot boot loader used by many Arduinos. A subset of the STK500 protocol is used for communicating with it. A factory pattern could be utilized to request a protocol capable of communicating with other boot loaders or full programmers (a separate chip doing the programming, requiring a full implementation of either v1 or v2 of STK500).

9 | Evaluation

This chapter covers the evaluation of the final product, the process model and what the team learned in this course.

9.1 Challenges

During the project several challenges were encountered. This section focuses on the two biggest of these challenges, licensing and the STK500 protocol.

9.1.1 Licensing

None of the group members had any experience with licensing prior to this project. The customer advised the group to use as much open source material as possible. All relevant open source systems deemed useful were incompatible with the Apache 2.0 licence, as they were licensed with GPL. Therefore it was necessary to implement the required functionality as part of the project. Initially the protocol was not a part of the assignment, but proved to be the most time consuming task of the project.

As there were insecurities regarding whether implementations of the STK500 protocol could be licensed outside of GPL, it had to be implemented as a library. Repeated attempts to get the matter clarified by Atmel, the copyright holder of the STK500 protocol, were unsuccessful.

9.1.2 Protocol

There were several difficulties related to programming the Arduino. There were few existing solutions for programming it in general, and none for Android or Java. The official implementation was available in C, but in the incompatible GPL license.

The existence of certain programs written in Java with the purpose of programming Arduinos, led the group into assuming one of these could be utilized, as the license was compatible. When the time came to make use of it, it was discovered to internally use AVRDUDE as well - an embedded version of the latter was available for several platforms.

This realization came somewhat late in the process, see Iteration 4 in Chapter B.4. Following this and the emergency meeting with the customer, it was decided to implement the protocol for programming Arduinos. Unfortunately, the documentation for these was a bit confusing; the most detailed and professional protocol document turned out to describe version 2 of the protocol [13]. Version 2 was incompatible with the bootloader installed on the Arduino by default (Optiboot v4.4 on Arduino Uno). This was also

the version that was partially implemented before the protocol differences and lack of backwards compatibility was discovered.

The incompatibility could have been discovered earlier, but testing of the protocol had to be postponed. This was due to a lack of available working serial communication libraries in Java for PC's compatible with the Arduino Uno USB interface. Therefore a simple Android I/O app was added for testing purposes. Even then, the reason for the failure to communicate with the Arduino was hard to pin down, until a second Arduino was wired up to echo back the information the pair received.

Changing the bootloader to utilize version 2 of the protocol could not be done due to it being licensed under GPL. Some attempts were made at finding compiled bootloaders for the Uno, but did not bear fruit. Attempts at compiling bootloaders using version 2 for the chip failed due to unresolved dependencies in the bootloader projects discovered, like the Stk-Boot [14] and uOS-Embedded [15].

The solution was to implement version 1 of the protocol, and write off most of the protocol work as waste (a bit less than a weeks work). Some of the work done on version 2 could be reused despite the protocols being radically different, however. The second version of the protocol has a standardized message format with checksums and static tokens inside the messages to aid in error discovery; this would have been beneficial for communicating over Bluetooth.

It was known that Optiboot didn't make use of all the commands specified in the protocol document AVR061 [16], but documentation on which commands were superfluous was lacking. Since Optiboot would simply report that everything is fine when encountering such a command, discovering which were implemented or not had to be done by trial and error or investigating the source. The latter is designed to be very compact to fit in the 512kB threshold, which, if exceeded, would make the bootloader occupy 1024kB.

Communicating with the Arduino proved fairly straightforward, but dealing with false positives was an issue. The Arduino would report that writing took place successfully, but writing would, for example, occur only in partial sections of the memory areas requested. The only way erroneous writing could be discovered, was to afterwards use an external program to read the written memory to verify its content.

These issues prolonged the development of the system by a considerable amount, and had a major impact on the project. Some of these issues could have been avoided by more careful initial investigation, while others would not have been apparent prior to implementing the protocol.

9.1.3 Hard reset

Due the development of the protocol, there were major problems programming the device without errors and it was needed a way to reset the Arduino to recover the programming. A lot of research were done to find out how the Arduino could be reset without using the excisting soft reset (using the ComputerSerial library).

The hard reset feature was intended to provide the final fall back; this would be the only way to automatically recover from serious communication problems during programming

(the Optiboot boot loader only waits for so long before leaving programming mode). In the case of the device ceasing to communicate with the protocol, reprogramming the device from a computer will be required in the absence of a hard reset feature.

9.2 Interaction with the customer

Regular meetings with the customer were scheduled every Friday throughout the project. These meetings were used to present status, progress and any problems that were encountered. In these meetings the customer was able to aid the group in decisions and give input on the prototypes of the product. Apart from these regular meetings, the communication with the customer was done by e-mail. If difficult problems were encountered, or difficult decisions had to be made, the group was always able to achieve help from the customer. Extraordinary meetings could be scheduled at short notice if needed, and the customer was willing to request help from external developers if the group requested it.

9.3 Group interaction

It was decided to schedule three regular group meetings each week, in addition to the meeting with the customer every Friday. Among these, two were to be team orientation meetings, where the group could discuss progress and issues and make plans for the coming week. The third meeting was a working session where the group worked together the whole day. In addition to these regular meetings the group scheduled working sessions continuously. The guidelines for the project stated that each group member should work approximately 20 hours each week, which meant at least three full days of work per group member each week. As described in Chapter 3.3, interaction outside group meetings was mainly done by e-mail or on Skype. In Table 9.1 the total work hours for the group on every iteration is listed. The reduction of hours in iteration 4 was due to the Easter holidays.

Regarding interaction between the group members, no major issues were encountered. All team members were minded to work at least the suggested amount of hours each week and deliver a finished product at delivery date. To prevent potential conflicts, rules and roles were established at an early stage of the project. By establishing this at the beginning of the project, the group might have avoided larger discussions and disagreements at a later stage. Based on this, it is believed that the preventive measures taken by the group in the initiation phase has played a major role in the avoidance of conflicts and disagreements throughout the project.

Table 9.1: Iterations and time spent

Iteration	**Hours**
Iteration 1 (Week 6-7)	186 hours
Iteration 2 (Week 8-9)	257 hours
Iteration 3 (Week 10-11)	233 hours
Iteration 4 (Week 12-13)	129 hours
Iteration 5 (Week 14-15)	195 hours
Iteration 6 (Week 16-17)	248.5 hours
Iteration 7 (Week 18-19)	438 hours

An associated chart diagram of the work hours is shown in Figure 9.1

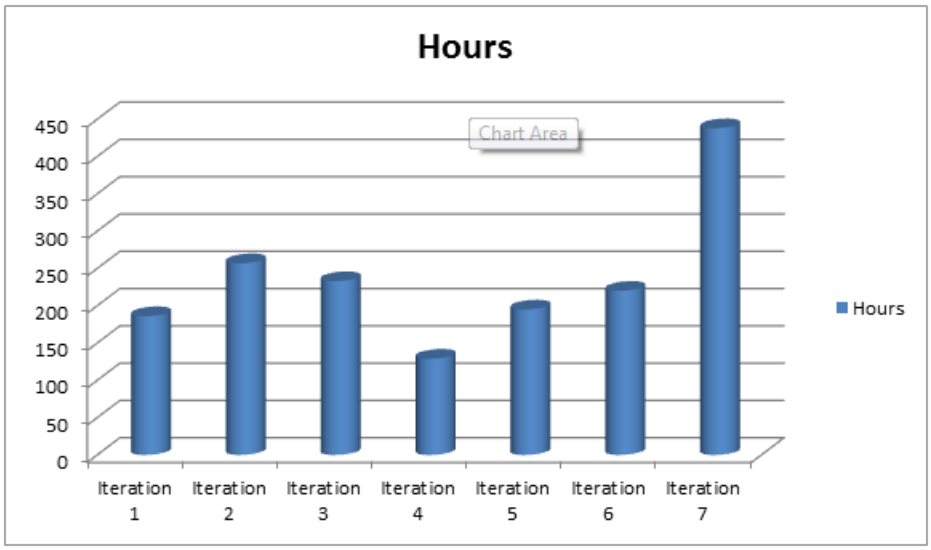


Figure 9.1: Work hours in each iteration throughout this project

9.4 Lessons learned

This section describes the most important lessons learned by the group throughout the project.

9.4.1 Prestudy

As mentioned in Section 9.1.2, it was originally assumed that an existing implementation of the STK500 protocol could be utilized. A more thorough prestudy would have revealed this assumption to be false at a considerably earlier stage. This would have allowed for more time to plan, implement, and test the implementation of the protocol. The lesson learned from this is to do a thorough prestudy early in the project, to back up every assumption with facts.

9.4.2 Project management

Based on the fact that every member of the group were used to different methods of communications during projects, the communication exceeded the expectations. This was because all members regularly checked Skype channel for messages. The lesson learned was that Skype proved to be an effective channel of communication within the group.

As described in Chapter 5.1.2.2 GitHub was used for version control. It is also possible to create issues highlighting what needs to be done within a project. These issues can be assigned to people, given labels to describe the issue, and be grouped to create a milestone. At times errors that should have been spotted have been merged into the master branch, introducing these errors to the master branch. These errors can be difficult to notice after the merging. An example of this is creating new features based on erroneous foundations. Based on this, future projects should meet test requirements before being merged into the master branch.

9.4.3 Design and Implementation

As described in Chapter 6.2.1, a complete design guide for the Android application was created early in the project. This guide proved to be very useful once the implementation began, as discussions on the design were avoided. Based on this, the group has learned that thorough planning of implementation is wise, as it increases the speed of the process and improves the quality of the final product.

9.4.4 Testing

Debugging the implementation of the STK500v1 protocol proved to be gruelling work. As described in Chapter 3.1.1, testing was less prioritized than the implementation. As none of the group members had significant experience on the field, the writing of tests could potentially demand a lot of time that was used on other tasks. In hindsight, continuously writing tests for each new module in both the Android application and the protocol implementation, might have saved the group a lot of time spent on debugging.

9.5 Conclusion

The most important requirements were met. Those requirements not met were less prioritized after discussions with the customer. The project management methods applied, ensured steady progress throughout the project. The developer team considered the project to be successful as the single most important goal was attained: over-the-air installation.

List of Tables

3.1 Roles 15
3.2 Work Breakdown Structure 17
3.4 Risk analysis 20

4.1 Use case 1 25
4.2 Use case 2 26
4.3 Use case 3 28
4.4 Use case 4 30
4.5 Use case 5 31
4.6 Use case 6 32
4.7 Use case 7 33
4.8 Use case 8 34

7.1 Connect with device using device list 60
7.2 Connect with device using QR Code 60
7.3 Connect with device using serial 61
7.4 Search for desired app 61
7.5 Connect to last connected device on application startup 62
7.6 Browse apps test 62
7.7 Install application on Arduino device 63
7.8 Change connected device 63
7.9 System test results 64
7.11 Functional requirements 65

9.1 Iterations and time spent 73

C.1 Example of component information XML document 88
C.2 Example of pin-setup-XML document 88
C.3 Serial number to connected microcontroller 89
C.4 The XML document for current connected device 89
C.5 he XML document for an example μC application. The Warn or require field specify whether or not the component has to be present or if the application can run without. 89
C.6 Comparison between two XML objects 90

List of Figures

1.1 Customer: 1.1(a) SINTEF, 1.1(b) NTNU. 2

2.1 Rational Unified process Model 7
2.2 Google Play 10
2.3 Pebble watch 11
2.4 Wiring Diagram for BT module 12

3.1 Gant chart 16
3.2 WBS tree showing a more detailed overview of the different tasks in each of the five milestones. 19

4.1 Use case 1 24
4.2 Use case 2 25
4.3 Use case 3 27
4.4 Use case 4 29
4.5 Use case 5 30
4.6 Use case 6 31
4.7 Use case 7 32
4.8 Use case 8 33

6.1 BT connection in service 40
6.2 Over The Air Architecture 41
6.3 System Architecture 42
6.4 UML - AddDeviceScreen 42
6.5 UML - Categories 43
6.6 UML - Main GUI 44
6.7 UML - Protocol 45
6.8 Sequence diagram for programming 46
6.9 Screen 1a - Device list 48
6.10 Screen 1b - Add device manually 49
6.11 Screen 1b-i - Input serial 49
6.12 Screen 2a - Browse shop 50
6.13 Screen 2b - Browse shop by category 50
6.14 Screen 3a - Application view 51
6.15 Screen 3a-i - Installation confirmation 52
6.16 Screen 3a-ii - Progress of installation 53
6.17 Screen Xa - Action overflow 53
6.18 UML - Welcome Screen 55
6.19 ER Diagram 56

8.1 How to wire the hard reset using the HC-05 [7]. 69

9.1 Work hours in each iteration throughout this project 74

A.1 Example of activity plan for the weeks 6 and 7 82

C.1 Example of a wiring diagram (scheme) 88

D.1 The welcome screen of the μC Software Store. 92
D.2 D.2(a) The device list with Bluetooth devices. D.2(b) The device list when connected to a device. 93
D.3 The options for adding devices 94
D.4 D.4(a) The device list with Bluetooth devices. D.4(b) Screenshot from the shop with all apps shown. 95
D.5 App View showing information about a single app 96

E.1 Change settings in the serial monitor in the Arduino IDE to configure the Bluetooth module. 99
E.2 Standard wiring of Bluetooth module: E.2(a). To configure the Bluetooth module through the Arduino, wire the module like this: E.2(b). 100

Bibliography

[1] *electronicdesign.com* N.P.. Web. May 01 2013 <http://electronicdesign.com/communications/what-s-difference-between-bit-rate-and-baud-rate>

[2] Kroll, P., and P. Kruchten. *The rational unified process made easy: A practitioner's guide to the rup.* Boston, MA: Addison-Wesley Professional, 2003. eBook.

[3] *Scrum.org.* N.P.. Web. Mar 13 2013 <http://www.scrum.org/Resources/What-is-Scrum>.

[4] Sommerville, Ian. *Software Engineering.* 9th ed. Boston: Pearson Custom Publishing, 2011. Print.

[5] Poppendieck, M., and T. Poppendieck. *Lean software development, an agile toolkit.* Addison-Wesley Professional, 2003. Print.

[6] *Sparkfun.com* N.P.. Web. Feb 21 2013 <https://www.sparkfun.com/products/10393>.

[7] *Iteadstudio.com* N.P.. Web. Feb 21 2013 <ftp://imall.iteadstudio.com/Modules/IM120723009/DS_IM120723009.pdf>.

[8] *Microsoft.com.* N.P.. Web. Mar 13 2013 <http://msdn.microsoft.com/en-us/library/aa292191(v=vs.71).aspx>

[9] *Microsoft.com.* N.P.. Web. Mar 13 2013 <http://msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx>

[10] *GeoSoft.no.* N.P.. Web. Mar 13 2013 <http://geosoft.no/development/unittesting.html>

[11] *Android.com.* N.P.. Web. Mar 13 2013 <http://developer.android.com/tools/testing/testing_android.html>

[12] Nielsen, Jakob. "Why you only need to test with 5 users." *NNgroup.com.* N.P., Mar 19 2000. Web. 16 Apr 2013. <http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/>.

[13] *Atmel.com.* N.P.. Web. Feb 8 2013 <http://www.atmel.com/Images/doc2591.pdf>

[14] *Google.com* N.P.. Web. Mar 25 2013 <https://code.google.com/p/stkboot/>

[15] *Google.com* N.P.. Web. Mar 25 2013 <https://code.google.com/p/uos-embedded/>

[16] *Atmel.com.* N.P.. Web. Mar 26 2013 <http://www.atmel.com/Images/doc2525.pdf>

[17] *Buzzle.com.* N.P.. Web. Mar13 2013 <http://www.buzzle.com/articles/validation-testing.html>

[18] *Microsoft.com.* N.P.. Web. Mar 13 2013 <http://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx>

[19] *Wikipedia.org.* N.P.. Web. Mar 13 2013 <http://en.wikipedia.org/wiki/Functional_testing>

[20] *Android.com* N.P.. Web. May 5 2013 <http://developer.android.com/design>

[21] *http://git-scm.com/* N.P.. Web. May 12 2013 <http://git-scm.com/book/en/

Git-Branching-What-a-Branch-Is>
[22] *http://www.websequencediagrams.com/* N.P.. Web. May 12 2013 <http://www.websequencediagrams.com/>
[23] *Arduino.cc* N.P.. Web. Feb 6 2013 <http://arduino.cc/en/Main/Software>
[24] *Github.com* N.P.. Web. May 9 2013 <https://github.com/Prosjekt2-09arduino/ArduinoStore/tree/master/Arduino/ComputerSerial/>
[25] *Arduino.cc* N.P.. Web. Mar 24 2013 <http://arduino.cc/en/Guide/Libraries/>
[26] *Sparkfun.com* N.P.. Web. Feb 28 2013 <https://www.sparkfun.com/datasheets/Wireless/Bluetooth/rn-bluetooth-um.pdf>
[27] *Elecfreaks.com* N.P.. Web. Feb 28 2013 <http://elecfreaks.com/store/download/datasheet/Bluetooth/HC-0305%20serail%20module%20AT%20commamd%20set%20201104%20revised.pdf>
[28] *Arduino.cc* N.P.. Web. Mar 24 2013 <http://arduino.cc/en/Tutorial/HomePage>

Source of images

This section containts the all the sources for all the images used in the report. The images and sources that are not listed here is made by the project team.

Figure 1.1(a) in section 1.3 :
N.P.. Web. Apr 19 2013
<http://www.sintef.no>

Figure 1.1(b) in section 1.3 :
N.P.. Web. Apr 19 2013
<http://www.ntnu.no>

Figure 2.2 in section 2.2.1
N.P.. Web. Mar 13 2013
<http://droidtrends.com/wp-content/uploads/2012/07/Google-Play-Store-APK-3.7.15.jpg>

Figure 2.3 in section 2.2.2 :
N.P.. Web. Apr 19 2013
<http://getpebble.com/>

Figure 2.4 in section 2.3 :
N.P.. Web. Apr 19 2013
<http://fritzing.org/>

Figure 2.1 in section 2.1.1
N.P.. Web. Apr 19 2013
<http://www.c3ns.com/c3nsservices_3.html>

A | Project documents

Documents produced in relation to the project will be presented in this section. Both status reports and activity plans were to be created and delivered to the supervisor every other week.

A.1 Status report example

Below is a copy of a status report made by the group for the supervisor. This status report encompasses both week 6 and 7.

1. Introduction
These two weeks were the first weeks of project work, since little could be done prior to meeting the customer. During the first week there was one member missing.

2. Progress summary
In this two-week period we have completed on average 68% of the work scheduled on the activity plan. We believe that we still are on schedule. The work that was done was in general related to the design of the final product and making sure we have all the required libraries, licenses and other factors that need to be in order before we start programming the actual product.

3. Open / closed problems

Open:
- New requirement: friendlier bluetooth device pairing
- Bluetooth connection: need to add a bluetooth module to the arduino
- Non-documented preexisting code
- Product naming

Closed:
- Lacking source code: Will not use the compiled program in question
- Licenses: separate out licensed components into separate projects

4. Planned work for the next period
For the next period we will have to complete the designs of the project and complete version 0.1 to show to the customer.

5. Updated risks analysis

- License incompatibility: impact has gone down from 8 to 7. Importance has gone down from 10 to 8. This is because of information from the customer regarding ways to circumvent this issue (as mentioned above).

A.2 Activity plan example

Below is an example of an activity plan made by the group for the supervisor. This activity plan spans weeks 6 and 7.

Activity plan week 06 and 07

15 Studiepoeng means every resource (team member) works approximately 20 hours a week.

			Planning				Follow-up		
Nr	Work Package[1]	Activity	Resource	Planned work (hrs)	Start	Finish	Actual work (hrs)	Status (% complete)	Comment
1	1.1	Project management	All	12	04.02.2013	15.02.2013	15	50%	
2	1.1.1	Meetings with customer	All	10	08.02.2013	08.02.2013	16	100%	one meeting every friday
3	1.3.7	Documentation (Work with the preliminary report)	Jeppe, Ståle, Robin, Nina, Bjørn	105	04.02.2013	08.02.2013	50	95%	
4	1.2.3	Meeting with supervisor	All	5	06.02.2013	06.02.2013	2	50%	
5	1.2.5.1	Research over the air	Ståle, Bjørn	40	04.02.2013	13.02.2013	50	25.00%	Difficulties with licenses and bluetooth connection
6	1.2.1	Requirement specification	All	5	04.02.2013	08.02.2013	5	50%	New requirements
7	1.2.2	Determine project team roles	All	4	04.02.2013	08.02.2013	4	100%	
8	1.1.2	Preparing a demonstration (mockup) for customer	All	30	11.02.2013	15.02.2013	15	100%	
9	1.2.5.2	Research libraries	All	30	04.02.2013	15.02.2013	20	30%	Not good documentation
10	1.3.2	Verify & validate user requirements	All	12	15.02.2013	15.02.2013	12	80%	New requirements
11									
12									
13									

Figure A.1: Example of activity plan for the weeks 6 and 7

B | Iterations

This section contains an overview and a summary of each iteration of this project. The project was split into iterations of two weeks, and the content is based on the activity plan and the status report that was delivered to the supervisor. In each iteration the use cases described in chapter 4.4 is shown to present what the group was working on.

B.1 Iteration 1

Week 6 and 7

B.1.1 Summary

These were the first two weeks of the project and the first group meeting was arranged. Meeting with the customer to get the requirement specification and understanding of the projects goal was done. Required libraries, licenses and other factors was taken in account before the group could start developing. No use cases were generated in this iteration, as basic research and planning had to be done before scenarios could be generated.

B.1.2 Overview

- Confirmed the understanding of the task
- Ideas for identifying the capabilities of an Arduino device and how to communicate these to the store client
- Investigate potential solutions to facilitate over the air installation of Arduino applications
- Started the design of the GUI for the Android application
- Researched on Apache licenses
- Meeting with customer

B.2 Iteration 2

Week 8 and 9

B.2.1 Summary

Architectural work and design of the user interface of the Android application. Positive feedback on the GUI was received from the customer. Design of use cases and further research was performed.

B.2.2 Overview

- Planned architectural design
- Designing of GUI
- Research on over the air implementation
- Research Ubicollab libraries
- Use iJacket - generic application, Bluetooth connection code
- Implemented a first sketch of the content provider
- Meeting with customer

B.3 Iteration 3

Week 10 and 11

B.3.1 Summary

Focus on Bluetooth connection and over the air implementation was prioritized these weeks. The application was taking form and the bugs were beginning to show up. Use cases 2, 3, 4 and 5 were in focus during this iteration. There were some issues with over-the-air because of incompatible licenses and much work was put into research. The Android application was estimated to be 50% complete.

B.3.2 Overview

- Researched Ubicollab libraries
- Finished design of the GUI
- Established a Bluetooth connection between Android and Arduino
- Implemented last draft of the content provider
- Meeting with customer

B.4 Iteration 4

Week 12 and 13

B.4.1 Summary

This iteration became critical when it was discovered that existing implementations of the STK500 protocol were either incompatible with the required Apache v.2 license or not compatible with Android. An emergency meeting with the customer took place and it was decided to implement the STK500 protocol in Java. Developing the Sync Adapter was removed from the requirements, and the focus was moved to implementing the protocol as a library. Use cases 2, 3 and 4 were in focus during this iteration.

B.4.2 Overview

- Emergency meeting with customer
  - Not possible to use existing solutions of STK500 protocol
  - Agreement on implementing the STK500 in Java ourselves as a library
- Researched STK500
- Unit test sketches
- Continued work on develop the Android application

B.5 Iteration 5

Week 14 and 15

B.5.1 Summary

Made good progress and much was done. The group was more optimistic about the implementation of the protocol and the development of the application was almost complete. The group was now split into two, where one group was writing the protocol, and the other group was writing Unit tests and polishing the last parts of the application. Use cases 4, 5 and 7 were in focus during this iteration.

B.5.2 Overview

- Added storage of hex-files to content provider
- Wrote Robotium unit tests
- Wrote STK500 protocol in Java/Android
- Finishing of the Android application except for the STK500 part
- Meeting with customer

B.6 Iteration 6

Week 16 and 17

B.6.1 Summary

Unexpected issues with programming the Arduino nearly caused the implementation to fail, requiring a meeting with the customer. Suggestions for alternate solutions were presented, and it was decided to work in parallel on some of them. The customer was to attempt to ask some experts in the field for some code review. Shortly after the meeting the problem was resolved.

B.6.2 Overview

- Meeting on protocol implementation problems
- Problem resolution
- Implement the STK500 to the Android application

B.7 Iteration 7

Week 18 and 19

B.7.1 Summary

These were the last two weeks scheduled for the project.
A new wrapper was created to solve the problem encountered in B.6. The final parts of the application were completed and the STK500 was implemented into the application as a library. A workshop with the customer at SINTEF was held, and an acceptance test and an usability test were performed. Most of the time was spent on completing the documentation.

B.7.2 Overview

- Workshop for SINTEF
  - Demonstrated our finish application
  - Acceptance testing
  - Usability testing
- Finished the STK500 protocol
- Final touch and finishing of the Android application
- Finishing documentation

C | Standard for defining μC Software Store compatible microcontroller-based devices

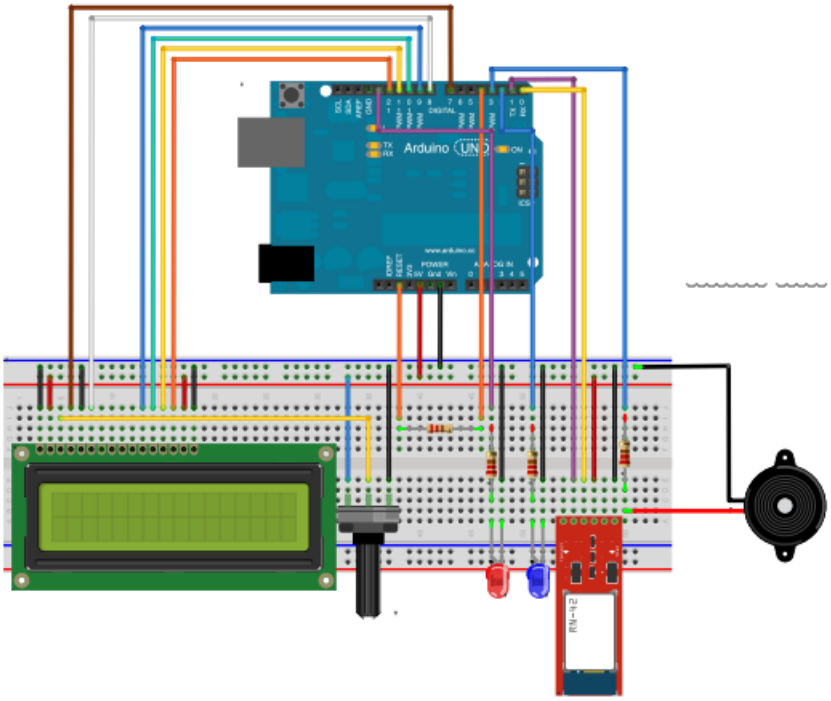
C.1 Introduction

This document contains a standard for how microcontroller-based devices is identified, and how this makes it possible to filter applications by compatible/incompatible in μC Software Store. It also defines certain requirements to the wiring of specific components.
All apps that will show up in the μC Software Store needs to be registered. This is done by sending an application to the management along with the microcontroller app, a wiring diagram and information about the components used.

C.2 Basics

When a μC-application is registered in the μC Software Store database, it will have a serial number on eight groups of four hexadecimal digits separated by colons, for example: *2F32:GSE2:F3S3:T32S:G443:45SN:K49D:03J2*. Every registered microcontroller also require having a wiring diagram (schema). Example in Figure C.1:

Figure C.1: Example of a wiring diagram (scheme)



All components used needs to be registered with the correct model number and the name of the producer, see Table C.1

Table C.1: Example of component information XML document

Category	**Model**	**Producer**
Screen	LCD-0025	SparkFun

In addition, a list of the component pins and which pins these are connected to on the device is required, like in the following example in Table C.2:

Table C.2: Example of pin-setup-XML document

Component pin	**Device pin**
0	6
1	7

C.3 Technical

The serial number will be stored in the microcontroller‘s EEPROM. When a bluetooth connection between the Android application and the microcontroller is established, the serial number is sent from the microcontrollers EEPROM to the Android application. The Android application will lookup the serial number towards a remote database, and retrieve an XML document with information about components and how the pins are connected. If the microcontroller does not have a serial number, or the serial number does not exist in the remote database, no application filtering can be performed.

This XML document will be converted to an object that is easier to handle. Every μC application in the database also has a corresponding XML document that is also converted to an object. This second XML document contains the requirements for the application; for each device mentioned, it will be specified whether this requirement is absolute or not. Apart from this they are identical in structure.

When the user clicks on "hide incompatible" the application will filter away (hide) the μC apps that have more features than the currently connected microcontroller can offer (if a conflicting pin-setup can't be automatically resolved, this will also happen).

C.4 Example

This is the connected device with serial number as shown in Table C.3:

Table C.3: Serial number to connected microcontroller

REGR:3T2A:345D:9999:FDHG:53FD:6Y5Y:3FDS

> Doing an XML lookup to remote database and retrieving the XML that belongs to the connected device. Table C.4 shows the XML for current connected device:

Table C.4: The XML document for current connected device

Category	**Model**	**Producer**
	Arduino Uno	Open Source
Screen	LCD-0025	SparkFun
Red LED	COM-09590	SparkFun
BT Module	T9J-RN42	SparkFun

These (Table C.5)are the requirements for an example μC application.

Table C.5: he XML document for an example μC application. The Warn or require field specify whether or not the component has to be present or if the application can run without.

Category	**Model**	**Producer**	**Warn or require**
	Arduino Uno	Open Source	
Screen	LCD-0025	SparkFun	Require
Red LED	COM-09590	SparkFun	Warn
Green LED	COM-09592	SparkFun	Warn
BT Module	T9J-RN42	SparkFun	Require
Speaker	COM-09151	SparkFun	Warn
Potentiometer	U-103	Sharma	Require

> The user selects "hide incompatible".

The system (Android application) is doing a check of which applications should be hidden/visible as illutrated in Table C.6:

Table C.6: Comparison between two XML objects

Connected Device		**Example μC app**	
Model	**Producer**	**Model**	**Producer**
Arduino Uno	Open Source	Arduino Uno	Open Source
LCD-0025	SparkFun	LCD-0025	SparkFun
COM-09590	SparkFun	COM-09590	SparkFun
T9J-RN42	SparkFun	T9J-RN42	SparkFun
COM-09592	SparkFun	?	?
COM-09151	SparkFun	?	?
U-103	Sharma	?	?

> Example μC application will be hidden because it requires more functionality than the current connected microcontroller can offer.

D | μCSS User manual

D.1 Introduction

This is the user manual for the μC Software Store application. It contains all the information needed regarding common use and options. μC Software Store is an application for the Android platform made as an app store containing applications for Arduino based projects. You can browse different applications for Arduino projects and install them through a Bluetooth connection. It is assumed that μC Software Store, and optionally a QR Code reader, is installed on an Android device and that a properly configured Arduino project with Bluetooth is nearby.

D.2 Welcome Screen

The first screen encountered when entering μC Software Store for the first time, is the welcome screen. This screen offers the choice between going directly to the shop to browse for apps or checking devices located in the area.

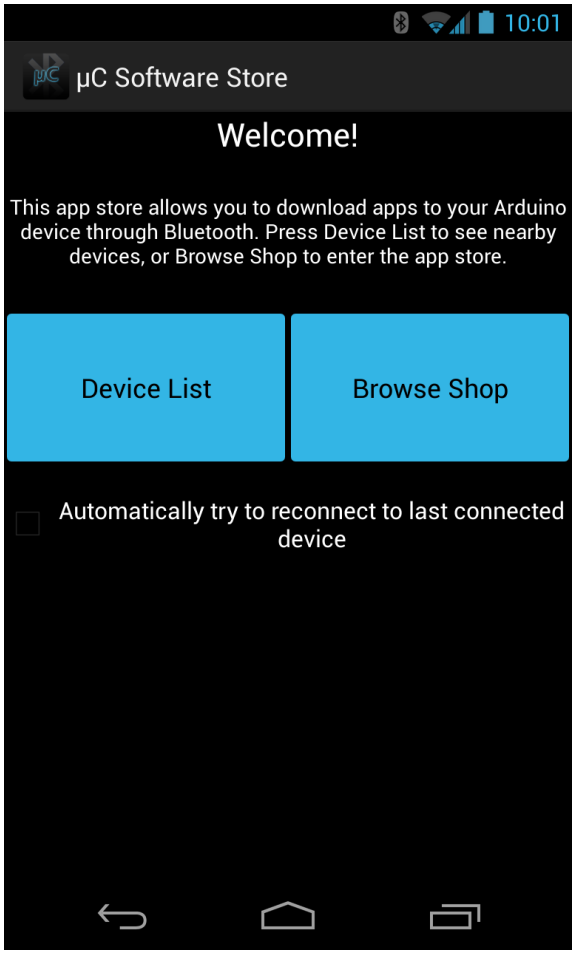
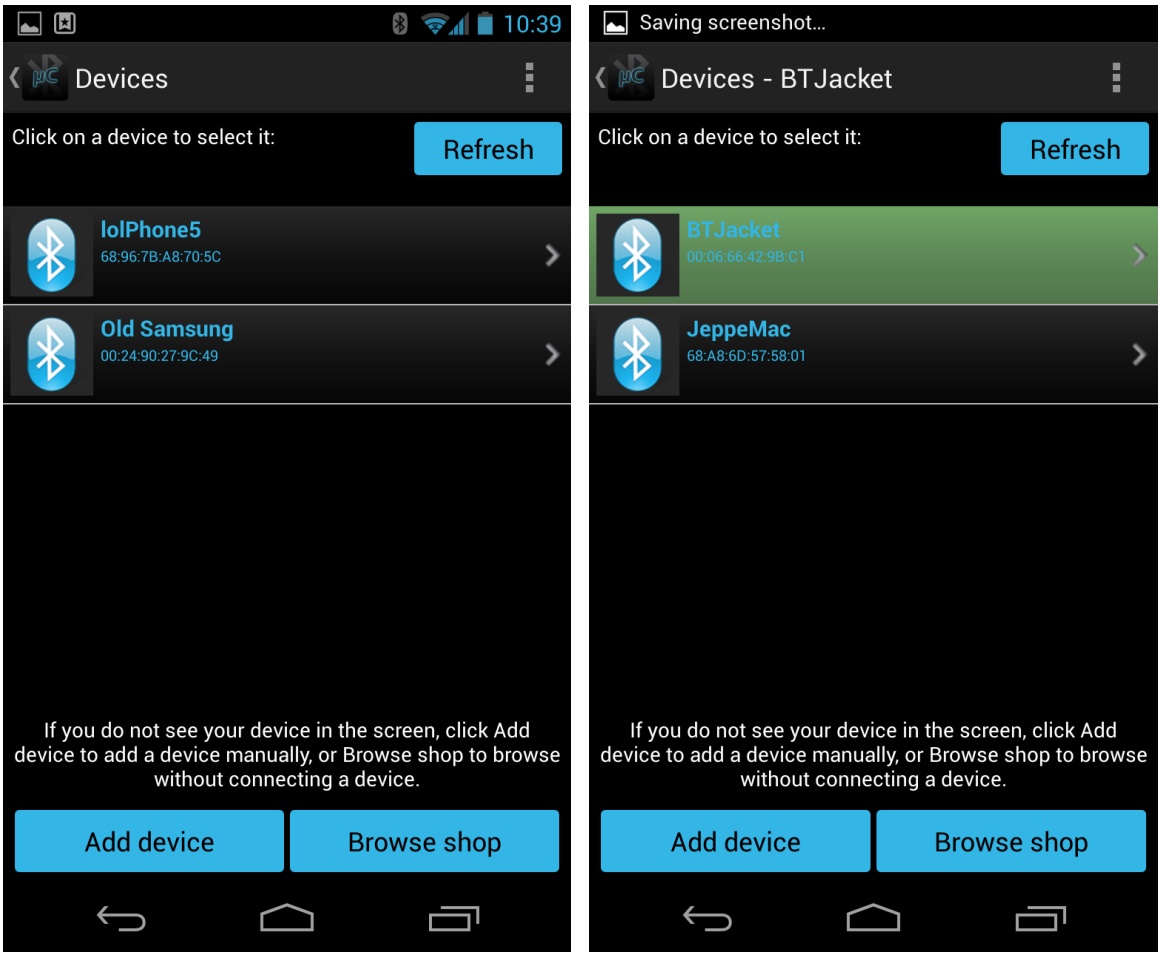

10:01
µC Software Store
Welcome!
This app store allows you to download apps to your Arduino device through Bluetooth. Press Device List to see nearby devices, or Browse Shop to enter the app store.
Device List
Browse Shop
Automatically try to reconnect to last connected device

Figure D.1: The welcome screen of the μC Software Store.

After the first time the application was launched, the welcome screen will attempt to automatically reconnect to the last known connected device. This is an option which can be turned off in the settings.

D.3 Device List

Upon selecting the "Device List" button from the welcome screen or from the action overflow, this screen will appear. On this screen, Bluetooth devices within the reach of the Android device will appear in a list. If the device you want to connect to doesn't show, or you do not want to connect to a device just yet, you can press either the "Add device" or "Browse shop" button respectively.


10:39
Devices
Click on a device to select it:
Refresh
lolPhone5
68:96:7B:A8:70:5C
Old Samsung
00:24:90:27:9C:49
If you do not see your device in the screen, click Add device to add a device manually, or Browse shop to browse without connecting a device.
Add device
Browse shop
Saving screenshot...
Devices - BTJacket
Click on a device to select it:
Refresh
BTJacket
00:06:66:42:9B:C1
JeppeMac
68:A8:6D:57:58:01
If you do not see your device in the screen, click Add device to add a device manually, or Browse shop to browse without connecting a device.
Add device
Browse shop

(a) (b)

Figure D.2: D.2(a) The device list with Bluetooth devices. D.2(b) The device list when connected to a device.

To connect to a device, simply press the appropriate device in the list and a connection will be established. Some devices require a password in order for connection to be made. In this case will selecting this device trigger a password input prompt. A dialog box will show whether or not the connection was sucecssfully established. The background of the now connected device in the list will turn green to indicate that it is connected.

D.3.1 Adding Device

Pressing the "Add device" button will give you a choice of how you want to add or connect to a device. You can attempt to connect to a device using either a QR Code or input a serial adress.

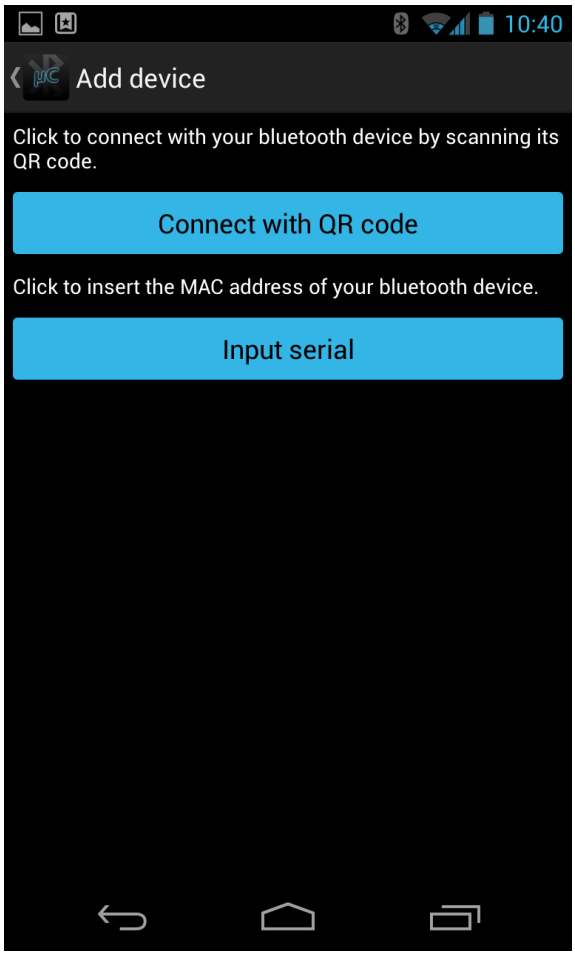

10:40
Add device
Click to connect with your bluetooth device by scanning its QR code.
Connect with QR code
Click to insert the MAC address of your bluetooth device.
Input serial

Figure D.3: The options for adding devices

D.3.1.1 QR Code

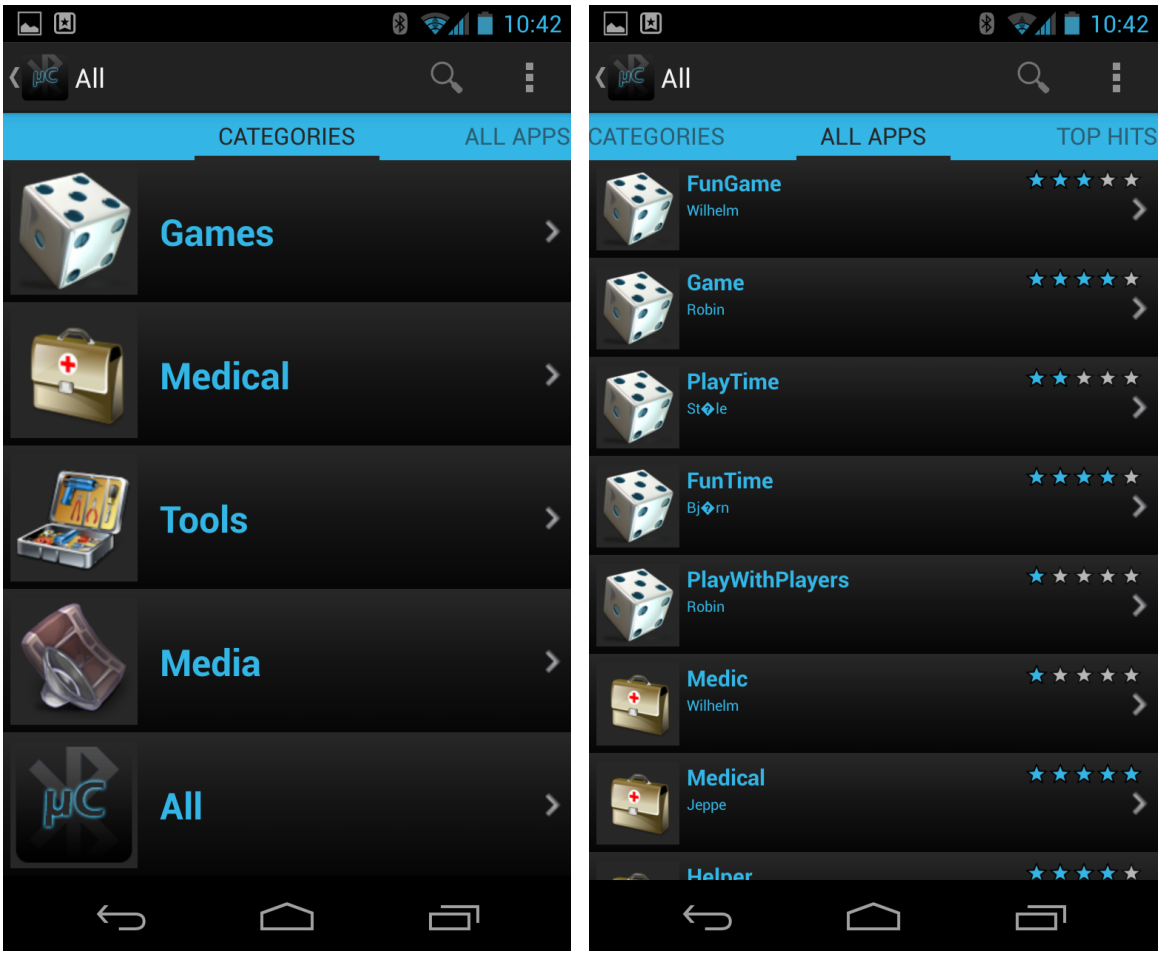
In order for you to connect with a QR Code, an app external app for scanning QR Codes must first be installed on the Android device. This will allow you to scan a QR Code which will automatically connect to an Arduino device if valid connection information is provided by the QR Code.

D.3.1.2 Serial

A connection can also be created by typing in a serial, which is the MAC address of the Arduino device. This serial is six groups of hexadecimal digits, meaning 0-9 and a-f, each group separated by a colon (:). You have to include the colon when typing this address, but it is not case sensitive which means you do not have to worry about capital letters.

D.4 The Shop

The first choice in the shop is the category screen.



(a) (b)

Figure D.4: D.4(a) The device list with Bluetooth devices. D.4(b) Screenshot from the shop with all apps shown.

You can filter through all of the apps in the store through these buttons. You can also browse all the apps together by pressing "All". There are two ways to sort the apps within the chosen category. Either not at all, which is the first screen that shows after selecting a category, or by the given rating. Each app within the category selected is now shown on the screen. The rating for each app, name of the app, name of the publisher, and the icon of the app is shown for all the apps. The user can go back to the categories by pressing the back button.

D.4.1 Searching

Wherever you are in the shop, you can start a search for an app you want. It doesn't matter how old or what buttons you have on your Android device, it is possible to start a

search from all buttons and devices. The search results will show after pressing the search button, and if there are no results you will be back at the shop screen with a toast telling you there were no results.

D.5 App View

Upon selecting an app in the shop, you will be taken to the app view. This view displays more information about the app you selected, like required hardware, pictures, developer and most importantly, an install button.

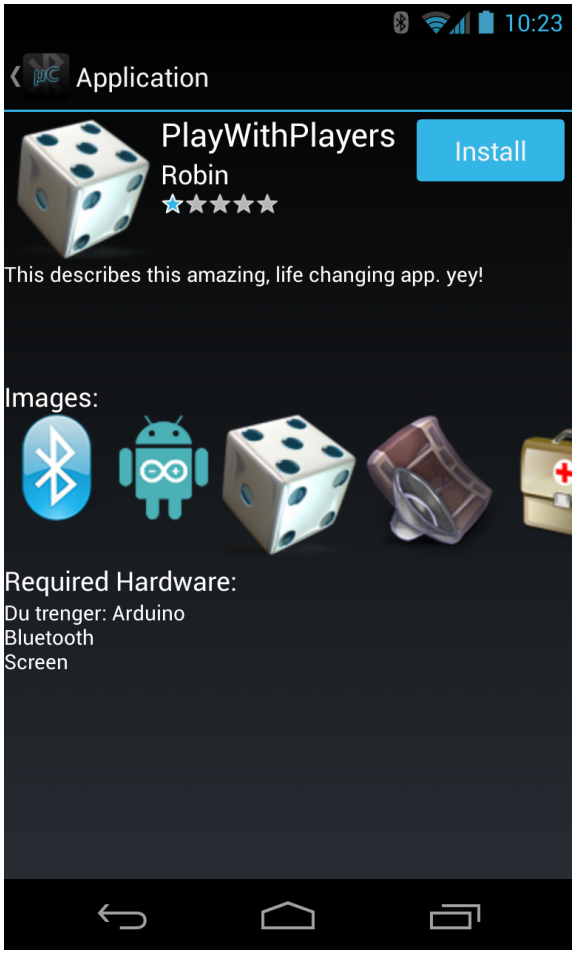


Figure D.5: App View showing information about a single app

Upon pressing the "Install" button, as long as you are connected to an Arduino device, you will be first asked to confirm the action. If you do the install will begin. If, however, you are not connected to a device, a dialog will open with the options of either going back to the app view, or go directly do the device list to connect to a device.

D.6 Settings

The settings available in μC Software Store are as follows:

Hide/Show incompatible
: Allows you to hide/show all apps not compatible with the currently connected device.

Connected device
: Shows the currently connected device.

Automatically reconnect
: Controls whether the application should attempt to automatically reconnect at the welcome screen when reentering the application.

Hide other devices
: Allows the option of hiding/showing devices in the device list which does not properly identifies themselves as an Arduino device.

E | Installation guide

This installation guide guides you through how to install apps on a new Arduino. If you use the Arduino provided with the report and Android application, it is not necessary to follow the first steps of this guide. If you are building your own Arduino, it is recommended to have programming expertise and know how serial communications works. You also need to be able to wire the Arduino by following a wiring diagram.

If you have bought a finished product, or are using the Arduino provided with the report, you can skip to Section E.2.

E.1 Upload the first application

Before you can use the μCSS, you need a standard application running on your Arduino, or else the Arduino will not be recognized as a supported device.

E.1.1 Arduino IDE

1. To transfer the first application to your Arduino, you need the Arduino IDE. Download the latest version of Arduino IDE [23] and install it.
2. Download the latest version of the ComputerSerial library [24] and import this into the Arduino IDE [25].
3. Plug a USB cable into your Arduino and make sure you do not have anything plugged into the digital port 0 and 1 on the Arduino, where the Bluetooth module usually is plugged into. This is because these pins are used for programming the device and if a component is wired to this pin, the device will not get programmed over USB.

E.1.2 Configure Bluetooth module

To program the Arduino over Bluetooth the Arduino requires a baud rate of 115200. Standard configuration of Bluetooth modules is usually configured to use a lower baud rate, therefore it has to reprogrammed to use the correct baud rate:

1. Upload an application that does not use the serial communication. The Arduino IDE comes with a lot of example applications and you can use one of them, like Blink. This can be found in the menu (File $\rightarrow$ Examples $\rightarrow$ Basic $\rightarrow$ Blink). Click the "Upload" button to upload.
2. Switch the wires from the Bluetooth module (TX on the Arduino to TX on the Bluetooth module, same with RX), as shown in Figure E.2(b). This is done because the Bluetooth module has to communicate with the computer through the USB cable. Remember to switch the wires back when you are finished configuring, as shown in Figure E.2(a).

3. Choose the right USB serial port from the menu.
4. Open the serial monitor.
5. Change from "Newline" to "Both NL & CR" and change baud rate to the baud of the Bluetooth module, as shown in Figure E.1. If you are not sure what baud rate you should use, you can try different baud rates until you find the right one.
6. You are now ready to send commands to the Bluetooth module.

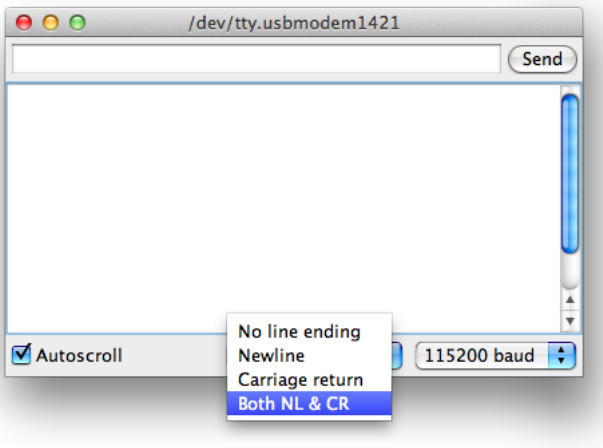
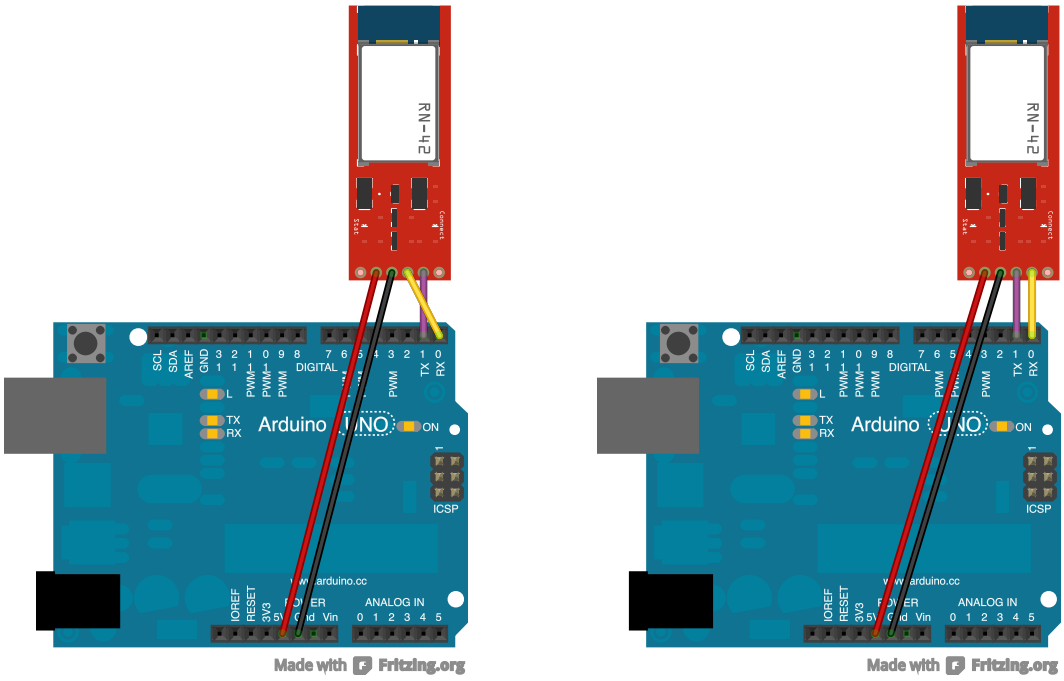


Figure E.1: Change settings in the serial monitor in the Arduino IDE to configure the Bluetooth module.



(a) (b)

Figure E.2: Standard wiring of Bluetooth module: E.2(a). To configure the Bluetooth module through the Arduino, wire the module like this: E.2(b).

E.1.3 Supported Bluetooth modules

You can use any Bluetooth module you want, but these are the modules that have been tested. Remember to power off the modules after they are reprogrammed to activate the new configuration.

When the Bluetooth module is programmed, turn the power off and on again to restart the module.

E.1.3.1 RN-42

To enter programming mode, type "$$$" and press enter. You should get "AOK" back when you entered programming mode. To change baud to 115200, write "SU,11" and press enter. You should now get "AOK" back.

Other supported commands can be found in the datasheet [26].

E.1.3.2 HC-05

Make sure you connect the KEY pin to 3.3v/5v (depending on which model of this module you have) before powering it up. This will allow you to enter the programming mode.

After you have successfully connected to the Bluetooth module, just write "AT" and press

enter. You should get "AOK" back when you entered programming mode. To change baud to 115200, type "AT+UART=115200,0,0". You should now get "AOK" back.

Other supported commands can be found in the datasheet [27].

E.1.4 Upload supported application

Open one of the example applications included in the ComputerSerial library using the Arduino IDE [23] (File → Examples → ComputerSerial → Examples). Upload by clicking the "Upload" button.

E.2 Android application

To be able to install the Android application on your device, you have to allow installation of applications from unknown sources on the device. On Android 4.x, this option can be found in the security settings on the device. In the security menu, check the box next to "Unknown sources".

To install the application on your device, you have to transfer the "uCSoftwareStore.apk" file to the device. This file can be found the provided CD. Once the file is transferred, press the file to install the application, and follow the on-screen instructions.

E.3 Recover

If the programming failed, the Arduino needs to be reconfigured from a computer.

1. Unplug the Bluetooth module.
2. Plug a USB cable from your computer to the Arduino.
3. Upload application as described in Section E.1.4.

E.4 Develop your own apps

To learn more about how the Arduino works, take a look at the tutorials on the official web page [28].

To make your applications work with the μCSS, remember to always import the ComputerSerial library. Take a look at the included examples in the ComputerSerial library to understand how it works.
Use of extended blocking commands (long running operations or loops inside the running loop) should be avoided, as these will prevent any programming during execution.